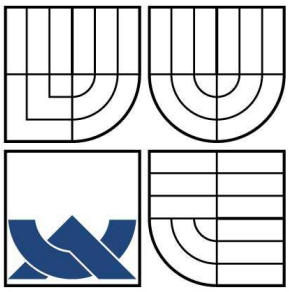
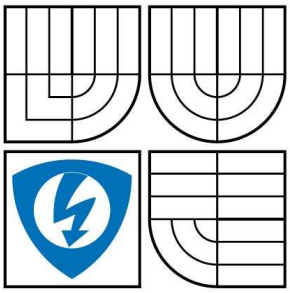


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKACNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

DÁLKOVÉ ŘÍZENÍ ZAPÍNÁNÍ INFORMAČNÍHO SYSTÉMU – ČÁST II

REMOTE - CONTROLLED SWITCHING OF INFORMATION SYSTEMS - PART II

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — DAVID PREČAN
AUTHOR

VEDOUCÍ PRÁCE — Ing. IVO HERMAN, CSc.
SUPERVISOR

BRNO 2008

ZDE VLOŽIT LIST ZADÁNÍ

(Kvůli správnému číslování)

ZDE VLOŽIT LIST LICENČNÍ SMLOUVY

(Kvůli správnému číslování)

ZDE VLOŽIT DRUHÝ LIST LICENČNÍ SMLOUVY

(Kvůli správnému číslování)

ABSTRAKT
Abstrakt práce v češtině

Práce se zabývá dálkovou komunikací prostřednictvím rozhraní Ethernet. Po stručné rešerši stávajících řešení je v práci navrženo dálkové měření teploty s přenosem dat prostřednictvím sítě Ethernet. Řešení je podrobně popsáno včetně konkrétního návrhu (HW i SW) zařízení pro přenos a vyhodnocení dat, využívající procesor ATmega 162/30. Zařízení dálkově komunikuje s PC, na kterém je zobrazována měřená teplota. Je popsán funkční vzorek, kterým byla ověřena správnost návrhu.

KLÍČOVÁ SLOVA
Klíčová slova v češtině

měření teploty, dálková komunikace, přenos dat, Ethernet, ATmega 162/30 procesor

ABSTRACT
Abstract in English

This bachelor´s thesis deals with remote communication over Ethernet interface. First, a brief survey of the existing solutions is presented. Then a remote temperature measurement using data transmission over Ethernet network is proposed. The proposed solution is described in detail including hardware and software design of an equipment for data transmission and evaluation that is based on ATmega 162/30 processor. This equipment communicates with a PC where the measured temperature is displayed. A functional sample of this equipment, which was used to verify the design, is described.

KEYWORDS
Keywords in English

temperature measurement, remote communication, data transmission, Ethernet, ATmega 162/30 processor

PREČAN D. *Dálkové řízení zapínání informačních systémů - část II*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 60 s., 10 s. příloh. Bakalářská práce. Vedoucí práce byl Ing. Ivo Herman, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma "Dálkové řízení zapínání informačních systémů - část II" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne ……. ……..

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu mé bakalářské práce Ing. Ivo Hermanovi, CSc. a panu Ing. Ladislavu Havlátovi za příkladnou metodickou, pedagogickou a odbornou pomoc a za další cenné rady při zpracování mé bakalářské práce.

V Brně dne …… ……..

(podpis autora)

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Cílem této bakalářské práce je navrhnout a po dohodě s vedoucím práce i realizovat dálkovou komunikaci prostřednictvím rozhraní Ethernet. V rámci práce navrhnout vhodnou konfiguraci konkrétního (hardware) zařízení, komunikačního protokolu a příslušného SW, který by umožňoval realizovat cíle zadání. Správnost návrhu ověřit na funkčním vzorku, který bude dálkově komunikovat s PC.

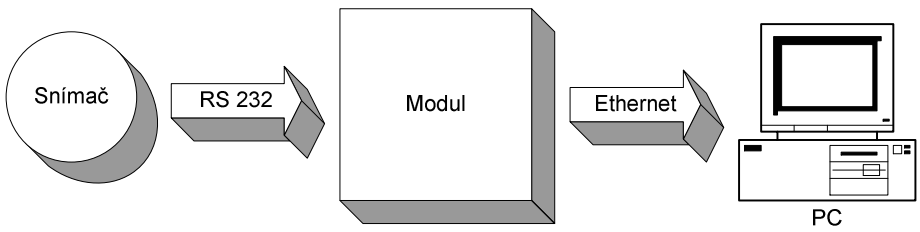
2 Výběr vhodného zařízení

2.1 Přehled současného stavu problematiky

V dostupné literatuře a internetu lze dnes nalézt řadu zařízení, která umožňují prostřednictvím dálkového přenosu dat snímat či vyhodnocovat různé měřené veličiny. Tato zařízení jsou dodávána jako hotové komponenty a využívají různé součástkové základny. Zařízení jsou většinou poměrně drahá, neboť umožňují realizovat celou řadu nejrůznějších funkcí. Do této skupiny patří například „Charon1“ (modul rozhraní Ethernetu) *[1].* Je rozhraním Ethernet - RS232 nebo RS485, lze jej použít dále jako WWW server, pro ovládání vstupů a výstupů, atd.
Charon I je vestavný modul s procesorem a řadičem Ethernetu RTL8019. Modul usnadňuje připojení libovolné aplikace do Ethernetu a Internetu bez složitého programování a čtení dokumentací. Charon I lze letovat přímo do plošného spoje základní desky aplikace, nebo jej osadit do patice. Tento modul je možno použít pro různé aplikace, například pro připojení čtečky čárového kódu, sériové tiskárny, LCD displeje nebo jakéhokoli sériového zařízení určeného pro připojení na port RS-232 (RS485).

2.2 Popis a rozbor řešení známého způsobu řešení

Zařízení popsaná v bodě 2.1 jsou většinou konstruována tak, že obsahují modul pro zpracování dat ze snímače a převod zpracovaných dat přes síťové rozhraní pomocí komunikačního protokolu do PC.



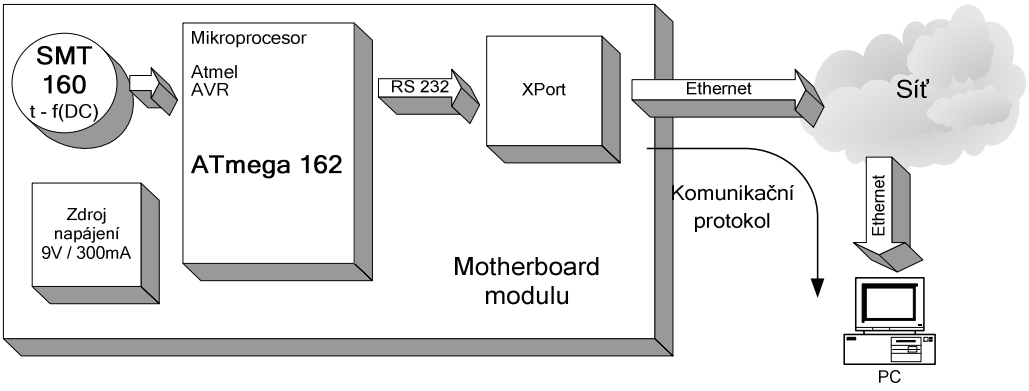
Obr. 1 Základní blokové schéma zařízení pro zpracování a přenos dat

Modul pro zpracování a převod dat je realizován pomocí mikroprocesoru, který je naprogramován tak, aby zpracovával data přijímaná ze snímače. Data zpracovaná procesorem jsou převáděna převodníkem z rozhraní RS 232 (422,

485) na Ethernet. Prostřednictvím komunikačního protokolu (TCP/IP) jsou data přenášena přes datovou síť do PC. Program v PC zpracovává přijatá data dle požadavků uživatele.
Modul je napájen ze samostatného zdroje, PC z elektrické sítě 230V stř.

3 Návrh vlastního řešení hardware

Na základě zadání jsem se rozhodl dálkově měřit teplotu v místě, kde je situován snímač teploty a měřenou hodnotu přenášet prostřednictvím internetu (případně jiné sítě) na PC. Měření teploty v místě snímače bude prováděno nepřetržitě. Při dálkovém připojení PC přes síť budou data zpracována na PC a zobrazena na monitoru PC pomocí příslušného softwaru. V souladu se zadáním jsem navrhl vlastní konstrukční řešení modulu pro zpracování dat ze snímače teploty a jejich přenos přes rozhraní Ethernet prostřednictvím komunikačního protokolu na PC. Při návrhu jsem vycházel z běžně používaného způsobu řešení.
Navržený modul je sestaven z následujících komponent (viz. Obr. 2)



Obr. 2 Blokové schéma řešení

- **SMT 160-30** - senzor teploty, pro jednoduchost je umístěn na společné desce tištěných spojů (motherboard modulu). Může však být umístěn mimo desku a připojen kabelem do vzdálenosti až 20m
- **ATmega162** - mikroprocesor AVR 8-Bit RISC
- **Zdroj napájení** 9V / 350mA
- **XPort** XE s úrovněmi TTL – převodník rozhranní Ethernet / RS 232
- **Síť internet** (intranet)
- **PC**, standardní osobní počítač se síťovou vstupní kartou

3.1 Senzor teploty SMT 160-30 *[2]*

3.1.1 Úvod a důvod použití

Senzor teploty SMT 160-30 jsem vybral kvůli tomu, že výstupem snímače je přímo obdélníkový signál o kmitočtu v rozmezí 1 – 4kHz, jehož střída se mění lineárně s teplotou. Střídou rozumíme poměr doby trvání úrovně log. “1” vůči celkové periodě. Další výhodou tohoto řešení je to, že lze měřit teplotu pouze s použitím jednoho vstupního pinu vyhodnocovacího zařízení. Snímač je již kalibrován při výrobě, takže odpadají problémy s jeho kalibrací. Pro měření venkovní teploty je postačující. Má široké využití v průmyslu a to od nejrůznějších domácích spotřebičů, klimatizací až po systémy vytápění.

3.1.2 Popis senzoru a jeho vlastností

Snímač má tři výstupní svorky. Dvě svorky pro napájení (5V) a třetí pro výstupní obdélníkový dvouhodnotový (logický) signál, který je převoditelný do číslicové podoby i bez použití A/D převodníku a je závislý na šířkové modulaci střídy impulsu.

Mezi jeho nejdůležitější vlastnosti patří :

- Napájecí napětí min. 4,75V a max. 7V. Optimální je **5V**
- Nevyžaduje A/D převodník
- Měří v rozsahu teplot od **-45 ºC do 130 ºC**
- Celková přesnost **0,7 ºC** je pro teplotní rozmezí (-30 ºC a 100 ºC) a **1,2 ºC** v rozmezí (-45 ºC do 130 ºC). Přesnost je definována jako maximální odchylka od závislosti popsané nominální rovnicí (uvedena níže, viz. výstupní signál). Respektuje všechny chyby. Platí pouze pro rozsahy teplot v trvalém provozu.
- Odchylka od linearity je závislá na teplotě. Pro rozmezí teplot (-30 ºC a 100 ºC) je menší než **0,2 ºC** (pro pouzdro TO-18). Nelinearita je definována jako odchylka od regresní přímky vypočtené z dat naměřených v celém rozsahu teplot
- Digitální výstupní signál je použitelný s logickými signály TTL, CMOS
- Snadné připojení přímo na pin procesoru
- Nízká spotřeba (< 1 mW)
- Cena obvodu je asi 100 Kč za jeden kus

3.1.3 Konstrukce senzoru

Základem integrovaného obvodu senzoru SMT 160 je polovodičový přechod PN polarizovaný v propustném směru společně s obvody převádějícími signál senzoru na modulaci střídy. Je vyroben na křemíkovém substrátu. Senzor je dodáván v různých pouzdrech typu :TO-92, TO-18, TO-220 a SOIC.

3.1.4 Výstupní signál

Střída impulsu, neboli poměr šířky impulsu k době periody se mění lineárně v závislosti na **teplotě** podle výrobcem stanovené rovnice :

$$DC = 0{,}32 + 0{,}0047 \cdot t \,. \qquad (3.1)$$

Kde : - **t** je teplota ve stupních celsia
- **DC** (duty cycle) je střída výstupního signálu o frekvenci 1 - 4 kHz.

3.2 Mikroprocesor AVR 8-Bit RISC - ATmega162 *[3], [19]*

3.2.1 Obecně o mikroprocesorech

Základním faktorem procesoru je počet bitů, které je schopen zpracovat v jednom kroku. Pro jednoduché aplikace se používají čtyřbitové nebo osmibitové procesory (Například osmibitový procesor umí počítat s čísly 0 až 255). Liší se také použitými obvody které jsou v něm obsaženy. Je to například velikost paměti FLASH, SRAM a EEPROM, počtem čítačů/časovačů a jejich rozlišením (např. 8 nebo 16bit), počtem portů, počtem sériových kanálů USART.
Dále pak mohou obsahovat analogový komparátor, obvod Watchdog (jestliže se cyklus nedostane na danou instrukci tak resetuje), některé i A/D převodník a spousty dalších obvodů. Detailní popis ke každému mikroprocesoru bývá uveden v materiálech od výrobce (datasheet).

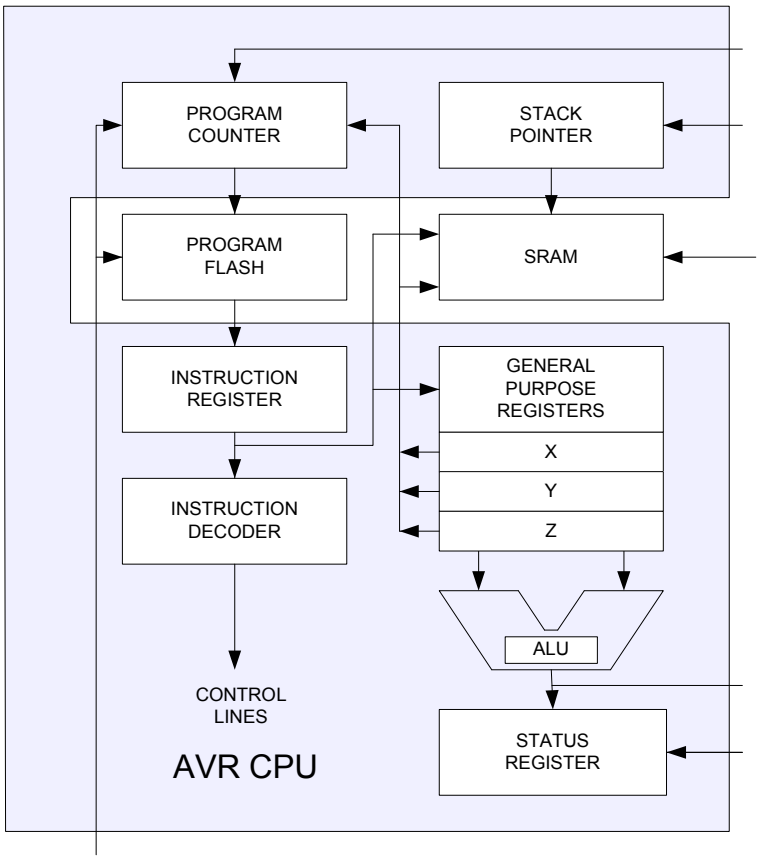
3.2.2 Princip a obvody mikroprocesoru

Mikroprocesor je logický integrovaný obvod. Program, který řídí mikroprocesor je uložen v paměti typu EEPROM. V průběhu programu tzv. čítač programu (program counter) vybere instrukci z paměti programu a přesune ji do dekodéru instrukcí. Podle typu instrukce provádí danou činnost (např. přesun dat, aritmeticko-logické operace, bitové operace, skoky atd.). Společně s vykonáváním instrukcí jsou prováděny i další činnosti nezávislé na programu, jako je kontrola napájení, přerušení, analogový komparátor, A/D převody, sériový kanál a další. Výsledky těchto činností mohou ovlivnit činnost programu, případně jej přerušit, popřípadě resetovat procesor.

Základní částí mikroprocesoru je **centrální procesorová jednotka (CPU).**

Obsahuje tyto základní části:

- **ALU** (aritmeticko-logická jednotka), která vykonává pomocí registrů většinu instrukcí (aritmetické, logické a bitové instrukce a podmíněné skoky)
- **PC** (program counter - čítač programu), tento registr obsahuje adresu právě prováděné instrukce v paměti programu.
- **ID** (instruction decoder - dekodér instrukcí), zjistí typ instrukce
- **SREG** (status register - stavový registr), který obsahuje 8 bitů příznaků, které se nastavují po provedení instrukcí a je možné se na ně odkazovat a dále podle výsledku programu s nimi pracovat
- **Sada registrů** (general purpose registers)
- **SP** (stack pointer - ukazatel vrcholu zásobníku), je ukazatel maxima (přetečení) zásobníku, což je tzv. paměť LIFO fyzicky umístěná v operační paměti **SRAM**, sloužící k zálohování dat. SP obsahuje adresu naposledy uložených dat.

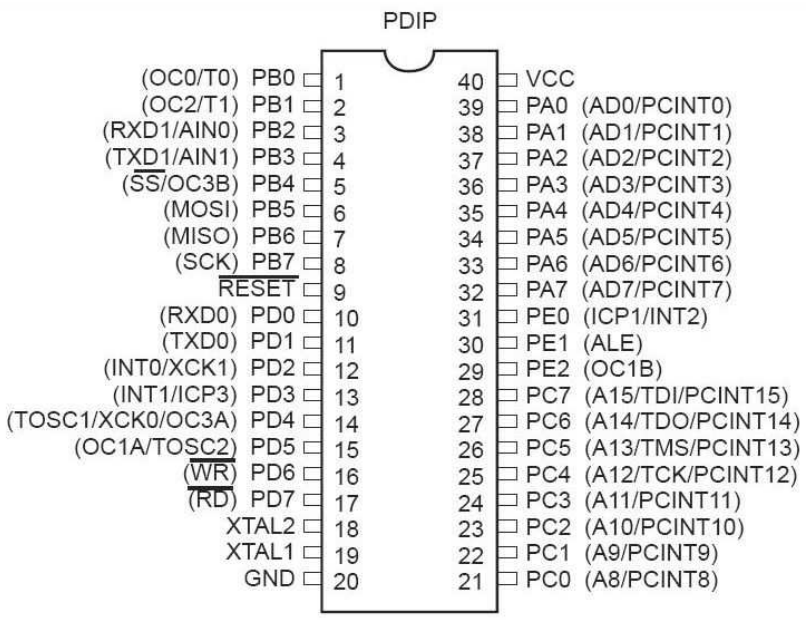


Obr. 3 Blokové schéma centrální procesorové jednotky

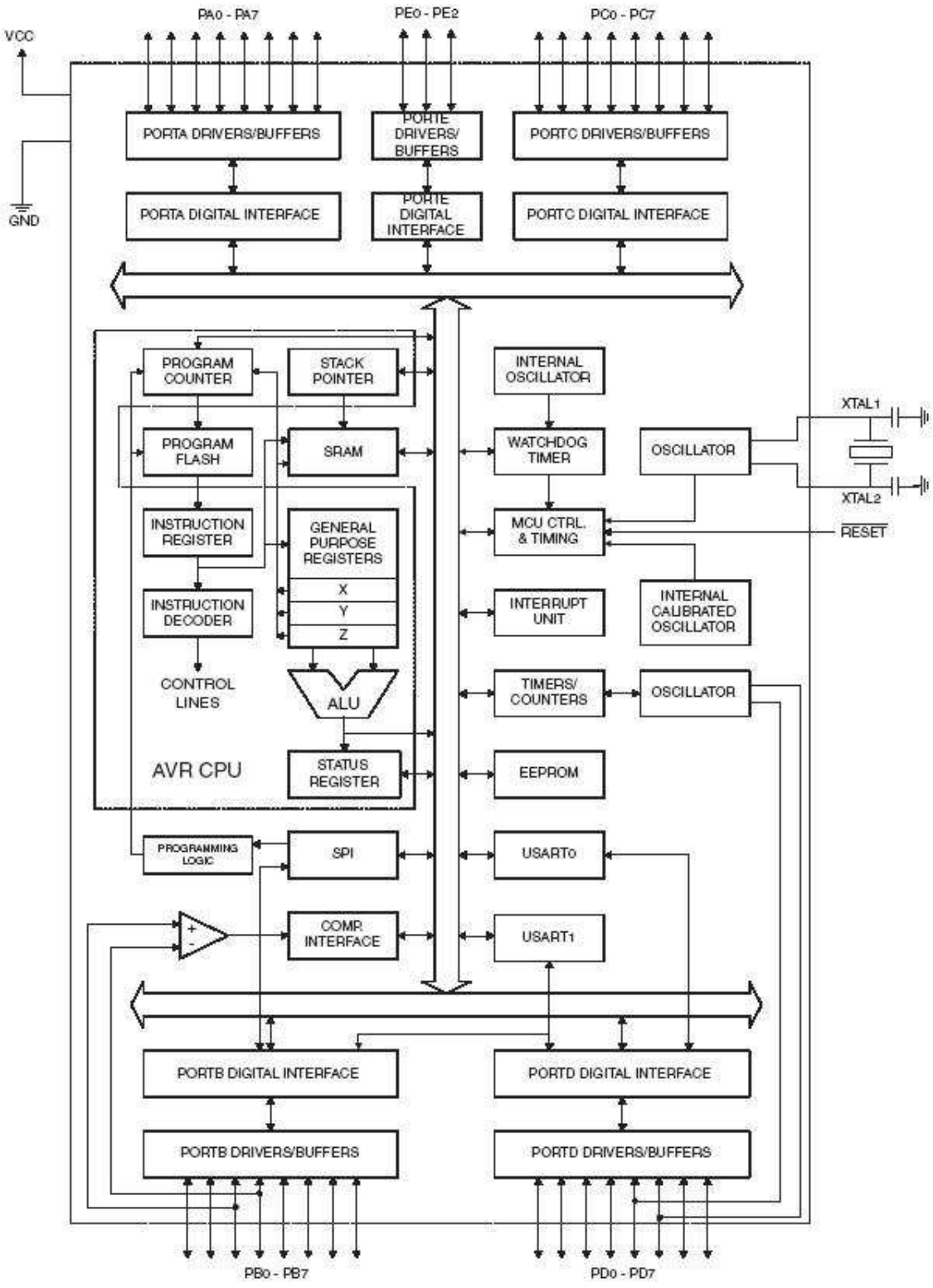
Aby mohl mikroprocesor komunikovat se svým okolím, jsou jeho součástí **I/O porty**, které jsou obousměrné a disponují ještě dalšími funkcemi.
Další důležitou součástí je **resetovací obvod**, který v případě nestandardního stavu (pokles napájecího napětí, impuls na resetovacím vstupu, watchdog timer) provede reset mikroprocesoru a program začíná probíhat znovu od adresy 0000h. Společně se začátkem programu se některé registry nastaví na definovanou úroveň.
Rychlost s jakou se vykonávají jednotlivé instrukce řídí **časovací obvody**, které jsou řízeny vnějším nebo vnitřním oscilátorem.

3.2.3 Mikroprocesor AVR 8-Bit RISC - ATmega162 *[3]*

ATmega162 je 8-bitový mikroprocesor založený na architektuře RISC (omezená instrukční sada, rychlé vykonávání instrukcí). ALU provádí většinu instrukcí ve spojení s 32 registry uloženými v paměti SRAM, z nichž 6 je organizováno jako 16-ti bitové registry (X,Y,Z). Paměť **SRAM** má kapacitu **1024 bytů**. Paměť dat **EEPROM** má kapacitu **512 bytů**.
S okolím mikroprocesor komunikuje pomocí **35 I/O** linek, u kterých je možno jednotlivě volit vstupní a výstupní režim. Všechny tyto linky mají ještě další funkce – analogový komparátor, 2x sériový kanál, 4x vstupy a výstupy čítačů/časovačů, vnější přerušení. Některé I/O linky mají i čtyři funkce. Dále mikroprocesor nabízí **dva osmibitové** a **dva šestnáctibitové čítače/časovače**. Časování může probíhat z vnitřního nebo vnějšího krystalu. **Reset** můžou vyvolat čtyři zdroje – vnější resetovací vstup, watchdog timer, power-on (připojení napájení), Brown-out (pokles napětí pod nastavenou úroveň). Napájecí napětí je 1,8 až 5,5V. Frekvence krystalu max. 16 MHz.



Obr. 4 Rozmístění pinů mikroprocesoru ATmega 162



Obr. 5 Blokové schéma obvodů mikroprocesoru ATmega 162

3.2.4 Programátor mikroprocesoru *[19]*

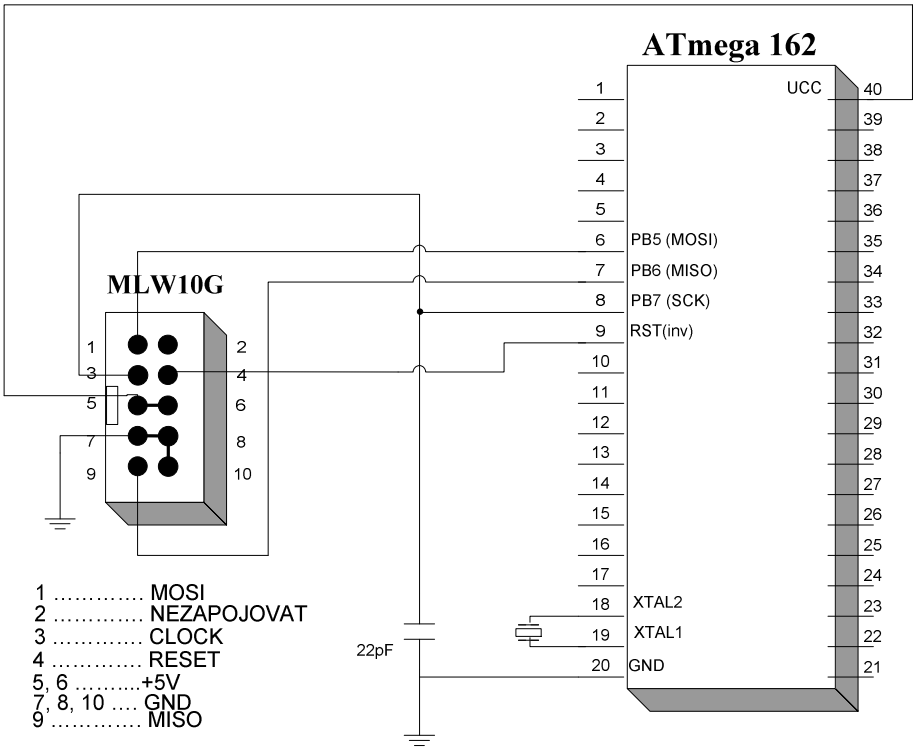
Mikrokontroléry AVR můžeme programovat buď paralelně (klasickým způsobem), nebo sériově (pomocí SPI rozhraní). Paralelní programování disponuje více možnostmi, je však komplikovanější. Programový mikrokontrolér je třeba naprogramovat v programátoru, potom vyjmout z programovací patice a vložit do zvolené aplikační desky. Tento postup komplikuje celý vývoj aplikace. Pro naše účely tedy zvolíme sériový download.

3.2.4.1 Sériový download [6], [19]

Sériový download je velmi pokroková metoda, umožňující programovat mikrokontrolér přímo na desce v aplikaci, čímž se celý vývoj aplikace výrazně urychlí (odpadá složité vkládání/vyjímání mikrokontroléru z/do patice programátoru).
Sériový download umožňují všechny mikrokontroléry z řady AVR (narozdíl třeba od řady AT89, u které je tento způsob programování spíše výjimečný), což byl jeden z důvodů použití našeho mikrokontroléru.
Jak programová, tak i datová paměť, může být programována použitím SPI sběrnice v okamžiku, kdy je RST(inv.) = 0 (převádí do stavu programování). Zapisovaná data jsou vzorkována náběžnou hranou SCK. Čtená data se objevují souběžně se sestupnou hranou SCK.

Sériové rozhraní obsahuje vývody : **SCK** (hodinový vstup), **RST(inv.)** (reset), **MOSI** (datový vstup), **MISO** (datový výstup).

Při programování musí být mezi vývody XTAL1 a XTAL2 připojen krystal. Na Obr. 6 je znázorněno zapojení SPI pro sériový download.



Obr. 6 Zapojení SPI pro sériový download

3.2.4.2 Postup při programování mikroprocesoru [6], [19]

Pro obsluhu programátoru využíváme vývojové prostředí WinAVR, které umožňuje volit rychlost přenosu dat od 9600 do 57600b/s. Řídící procesor si automaticky zjišťuje použitou rychlost. Jako standardní se však používá 9600b/s. Je-li přenos bez problémů, můžeme přenosovou rychlost zvýšit.

Postup při programování :

Zkontrolujeme, zda není vložen v žádné patici programátoru procesor. Plochým kabelem propojíme programátor s procesorem na naší desce a přivedeme napájecí napětí. Při správné identifikaci procesoru programátorem, blikne dvakrát červená led a poté se trvale rozsvítí žlutá (viz. Obr. 7). V opačném případě, (procesor nerozpoznán, popřípadě je vadný) bliká červená led trvale. Programátor je připraven k přenosu dat. Aplikační procesor se resetuje a naprogramuje. Během programování svítí červená led, která po úspěšném naprogramování zhasne a nový program automaticky spustí. Není třeba nic rozpojovat.

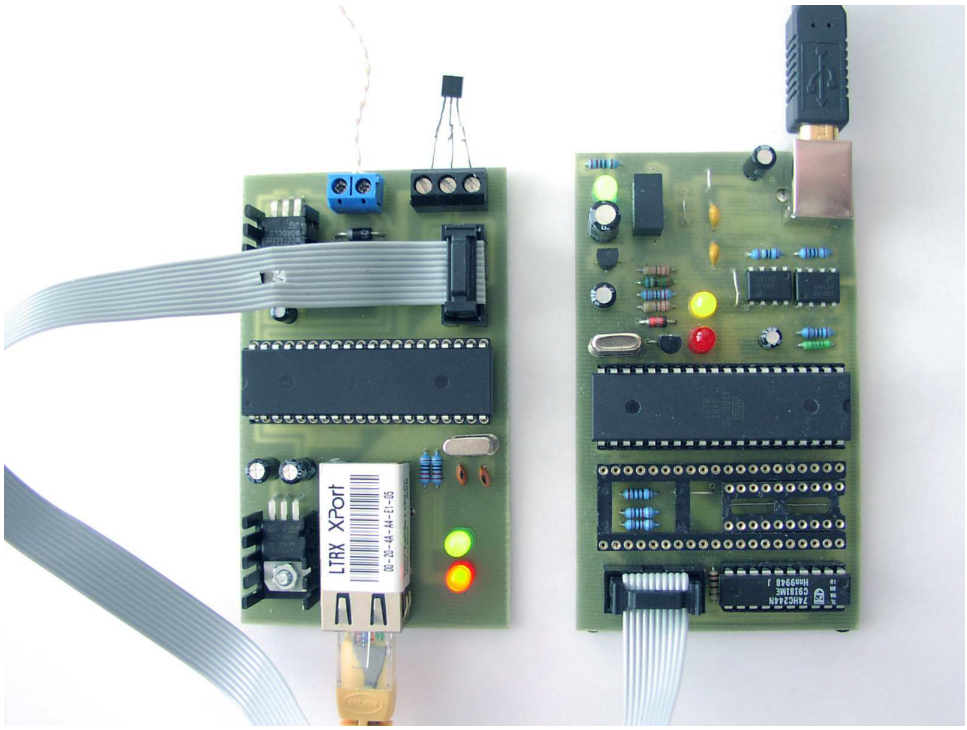
Pro správnou funkci rozhraní SPI musíme dodržet několik zásad :

- Propojovací kabel mezi programátorem a deskou zařízení musí být co nejkratší, doporučená délka je **do 25 cm**. Rychlost přenosu je velká a pravděpodobnost chyby roste s délkou kabelu.

- Na desce je třeba zapojit všechny **3 vodiče GND** aby bylo spojení zemí co nejlepší s co nejmenší indukčností, rovněž tak je třeba zapojit **oba vodiče +5V** (viz. Obr A).

- Mezi vývodem 3 (CLOCK) u konektoru MLW10G a GND je vhodné připojit **keramický kondenzátor 22pF** proti zemi pro potlačení zákmitů na vedení

- Během přenosu dat do procesoru se nesmí zapínat v okolí žádné elektrické zařízení. Rušení, které vzniká zapnutím nebo vypnutím může způsobit chybu přenosu dat.



Obr. 7 Sériové programování mikroprocesoru

Na obrázku vpravo je znázorněn programátor, použitý při programování mikrokontroléru (ve stavu, kdy identifikoval procesor) a vlevo je naše vývojová deska s procesorem.

3.2.5 Zpracování příchozího signálu ze snímače procesorem

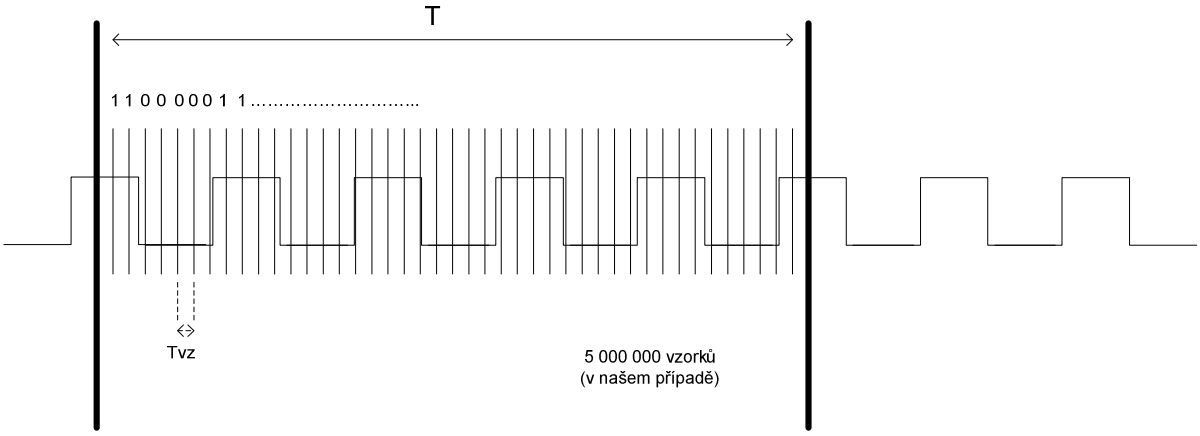
Jak jsem již výše uvedl, příchozí signál je obdélníkového typu a jeho střída se mění lineárně s teplotou. Tento signál přijde na port procesoru, který ho musí zpracovat (vyhodnotit). Procesor musí umět vykonat tyto dva základní kroky :

1) Zpracuje signál
2) Odesílá na XPort

3.2.5.1 Zpracování signálu

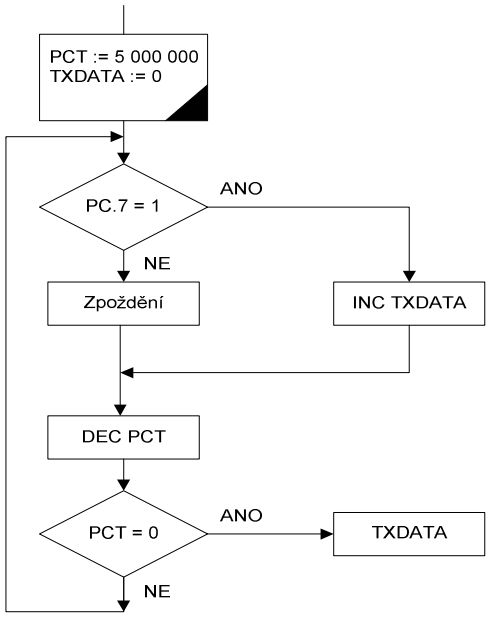
Aby bylo měření střídy **přesné** bez nutnosti synchronizace impulsů při každém měření, budu vzorkovat průběh signálu po určitou dobu periody T. Počet snímaných vzorků v dané periodě stanovím například 5 000 000. Čím větší počet vzorků budu snímat, tím větší dosáhnu přesnost měření, avšak měření bude trvat delší dobu. Každý vzorek mi bude určovat aktuální logickou úroveň signálu (tj. log. “0“ nebo log. “1“). Pro stanovení střídy (poměr doby trvání úrovně log. “1“ vůči celkové periodě) mě tedy bude zajímat počet log.“1“ v dané periodě, ze kterých jsem vypočítal střídu podle vztahu :

$$DC = \text{počet log.“1“} / 5\,000\,000 . \qquad (3.2)$$



Obr. 8 Vzorkování příchozího signálu ze snímače (zjednodušeně)

Program, který bude mikroprocesor vykonávat, bude postupovat podle algoritmu na Obr. 9, kde v proměnné (registru) TXDATA bude konečná hodnota log. “1“ při počtu snímaných vzorků v periodě PCT := 5 000 000.



Počáteční hodnota proměnné TXDATA je 0. Bude se inkrementovat (zvyšovat o 1) pokaždé, přijde-li na port např. PC.7 log. “1“. Přijde-li log. “0“ tak program pokračuje. Proměnná PCT (počet snímaných vzorků) se bude dekrementovat (snižovat o 1) do té doby, dokud nebude rovno nule. Při PCT = 0 je v proměnné TXDATA konečná hodnota log. “1“.

Obr. 9 Schématický postup programu mikroprocesoru při stanovení střídy

Konečná hodnota log. „1“ uložená v proměnné TXDATA je vysílána přes sériové rozhraní RS 232 na převodník Xport a odtud pomocí TCP/IP datagramů skrze datovou síť do PC. Program na straně příjemce (PC) vypočítá střídu dle výše uvedené rovnice.

Takto vypadá program pro výpočet vzorků s logickou hodnotou „1“ napsaný v assembleru :

V proměnné **PCT** je uložen počet snímaných vzorků (5 000 000), v proměnné **BSMT** (odpovídá příchozímu portu PC.7) je příchozí signál ze snímače (buď log. „0“ nebo log. „1“) a v **TXDATA** je konečná hodnota log. “1“ a počet snímaných prvků.

```
            CLR   R16               ; vynuluje registr
            CLR   R17
            CLR   R18
            CLR   R19


            LDI   R20, BYTE1(PCT)   ; nahraje přímo hodnotu PCT do
                                    ; registru
            LDI   R21, BYTE2(PCT)
            LDI   R22, BYTE3(PCT)
            LDI   R23, BYTE4(PCT)

            CLR   R0                ; vynuluje registr R0
LOOP:       CLC                     ; vynuluje se C (carry bit)
            SBIC  BSMT              ; přeskočí následující instrukci,
                                    ; je-li BSMT = 0
            SEC                     ; nastaví se C

            ADC   R16, R0;          ; sečte obsah dvou registrů včetně C
            ADC   R17, R0
            ADC   R18, R0
            ADC   R19, R0

            SUBI  R20, 1            ; odečítá z reg. R20 hodnotu 1
            SBCI  R21, 0            ; odečítá z registru konstantu a C
            SBCI  R22, 0
            SBCI  R23, 0
            BRNE  LOOP

            STS   TXDATA + 0, R16   ; do TXDATA se uloží naměřené hodnoty
            STS   TXDATA + 1, R17
            STS   TXDATA + 2, R18
            STS   TXDATA + 3, R19

            LDI   R16, BYTE1(PCT)   ; do TXDATA se uloží počet
                                    ; snímaných vzorků
            STS   TXDATA + 4, R16
            LDI   R16, BYTE2(PCT)
            STS   TXDATA + 5, R16
            LDI   R16, BYTE3(PCT)
            STS   TXDATA + 6, R16
            LDI   R16, BYTE4(PCT)
            STS   TXDATA + 7, R16
```

Pozn. Carry bit (C) bude nastaven, pokud se bude generovat přenos ze sedmého bitu (výsledek přeteče a „nevejde“ se do příslušného registru). V opačném případě je C vynulován. [19]

3.2.5.2 Odeslání dat na převodník XPort [3], [19]

Odeslání na převodník se děje pomocí sériových kanálů **USART**. Mikroprocesor ATmega162 využívá dva plně duplexní kanály pro sériovou komunikaci USART0 a USART1, umožňující pracovat ve dvou základních režimech :

- **Synchronní režim** (master – vysílá hodiny / slave – přijímá hodiny), kde TxD – vysílací bit, RxD – přijímací bit a XCK – hodiny (generátor hodinových pulsů)

- **Asynchronní režim**, má TxD – vysílací bit, RxD – přijímací bit

V našem případě bude využit asynchronní přenos.

Důležité registry, se kterými procesor pracuje při sériové komunikaci :

- **přijímací registr UDR** ke kterému máme přístup a můžeme z něj číst
- **sériový přijímací registr** do kterého se data během příjmu posouvají

A) Odeslání se provede zapsáním bytu do registru UDR. Proběhne-li vše v pořádku, v registru **UCSRA** se nastaví příznak **TCX**.

Takto vypadá hlavní část programu pro odeslání napsaná v assembleru :

```
OUT    UDR0, R18              ; vyšle byte v reg. R18
SBIS   UCSR0A, TXC            ; cekat na vyslání (při
                              ; vyprázdnění sériového
                              ; vysílacího registru TXC = 1)
RJMP   PC - 1

SBI    UCSR0A, TXC            ; vynulovat příznak TXC
```

Pozn. Do registru R18 se postupně načítají data (viz. Obr. 17 Blokové znázornění našeho protokolu), která se jednotlivě odesílají.

B) Přijetím znaku se nastaví příznak **RXC** v registru **UCSRA** a pak můžeme přijatý znak přečíst z registru **UDR** (má stejnou adresu jako při odeslání, ale fyzicky jsou to dva registry RXB a TXB). Do registru UDR se data přepisují až v okamžiku, kdy přijmeme celý byt a procesor detekuje startbit dalšího bytu. Přijmeme-li první byt, tak při startbitu dalšího (druhého bytu) se přepíše první byt do UDR. Při dalším startbitu třetího bytu se přepíše druhý byt do UDR atd. Do té doby musíme mít z registru UDR přečten první byt, jinak se přepíše druhým bytem. V případě, že dojde k přepsání, nastaví se chybový příznak DOR.

Pro odeslání a přijímání je nutné toto povolit bity **RXEN** a **TXEN**. Nepovolení má za následek nepřijetí dat a nastavení chybových bitů **FE, DOR** a **UPE.**

Registry, pomocí kterých <u>nastavujeme sériový přenos</u> :

1) <u>UCSRA</u>

Tab. 1 Nastavení registru UCSRA

Bit	7	6	5	4	3	2	1	0
	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM
Read/Write	R	R/W	R	R	R	R	R/W	R/W
Initial Value	0	0	1	0	0	0	0	0

Bit 7 **RXC** – příznak přijatého byte, nastaví se při dokončení příjmu a nuluje při čtení přijatých dat z registru UDR. Může generovat přerušení.

Bit 6 **TXC** – příznak odeslaného byte, nastaví se při dokončení vysílání (vyprázdnění sériového vysílacího registru) pokud nejsou v registru UDR zapsaná další data. Nuluje se automaticky při vyvolání přerušení nebo pomocí programu. Může generovat přerušení.

Bit 5 **UDRE** – příznak prázdného bufferu, nastaví se, je-li vysílací registr UDR prázdný (data jsou přesunuta do sériového vysílacího registru), připravený na zápis nových dat. Může generovat přerušení.

Bit 4 **FE** – chybový příznak, nastaví se v případě, že stopbit byl přijat jako log0

Bit 3 **DOR** – příznak přeplněného datového bufferu, nastaví se v případě, kdy přijímací registr nebyl přečten, druhý sériový přijímací registr je plný a byl detekován další startbit

Bit 2 **UPE** – chybový příznak, nastaví se v případě, že došlo k chybě parity chyba parity

Bit 1 **U2X** – využití při asynchronní komunikaci (při synchronní vždy log0) log1 nastaví dvojnásobnou přenosovou rychlost.

Bit 0 **MPCM** – povoluje multiprocesorovou komunikaci

2) <u>UCSRB</u>

Tab. 2 Nastavení registru UCSRB

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 **RXCIE** – povolení přerušení přijímače

Bit 6 **TXCIE** – povolení přerušení vysílače

Bit 5 **UDRIE** – povolení přerušení při vyprázdnění registru UDR

Bit 4 **RXEN** – povolení příjmu

Bit 3 **TXEN** – povolení vysílání

Bit 2 **UCSZ2** – společně s bity UCSZ1 a 0, zvolí počet datových bitů

Bit 1 **RXB8 –** devátý bit přijímacího registru

Bit 0 **TXB8 –** devátý bit vysílacího registru

3) UCSRC

Tab. 3 Nastavení registru UCSRC

Bit	7	6	5	4	3	2	1	0
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	1	0	0	0	0	1	1	0

Bit 7 **URSEL** - slouží k výběru registru UBRR/UCSRC. Tyto dva registry mají stejnou adresu a k rozlišení se kterým komunikujeme musíme dodržet následující pravidla:

- je-li bit URSEL = log0 – následujících 7 bitů se zapíše do registr UBRRH
- je-li bit URSEL = log1 – následujících 7 bitů se zapíše do registr UCSRC
- nebyl-li v předchozím strojovém cyklu čten tento registr, čteme UBRH
- byl-li v předchozím strojovém cyklu čten tento registr, čteme UCSRC

Bit 6 **UMSEL** – volí asynchronní nebo synchronní komunikaci

Bit 5, 4 **UPM1, UPM0** – mód parity (00) = bez parity, (01) = vyhrazená, (10) = sudá, (11) = lichá

Bit 3 **USBS** – volí počet stopbitů, (0) = jeden bit (1) = dva bity

Bit 2, 1 **UCSZ1, UCSZ0** – společně s UCZS2 volí počet datových bitů

Bit 0 **UCPOL** – polarita hodinového impulsu pro synchronní režim

4) UBRR

Registr určující přenosovou rychlost (baud rate). UBRR je složen ze dvou registrů: **UBBRH** (vyšší byte) a **UBRRL** (nižší byte).

Pro nastavení přenosové rychlosti musíme stanovit hodnotu UBRR pomocí rovnice (asynchronní normální mód, kde U2X = 1) :

$$\mathrm{UBBR} = \frac{\mathrm{f_{OSC}}}{16 \cdot \mathrm{BAUD}} - 1. \qquad (3.3)$$

Kde : - **BAUD** (Baud rate) je přenosová rychlost v bitech za sekundu (bps)
- $\mathbf{f_{osc}}$ je frekvence krystalu

V našem případě při použití vnějšího krystalu f_{osc} = 11,0592MHz a námi požadované přenosové rychlosti Baud rate = 9600 je vypočtená hodnota UBRR = 71 . Shoduje se s příkladovými hodnotami v tabulce v datasheetu procesoru *[3].*

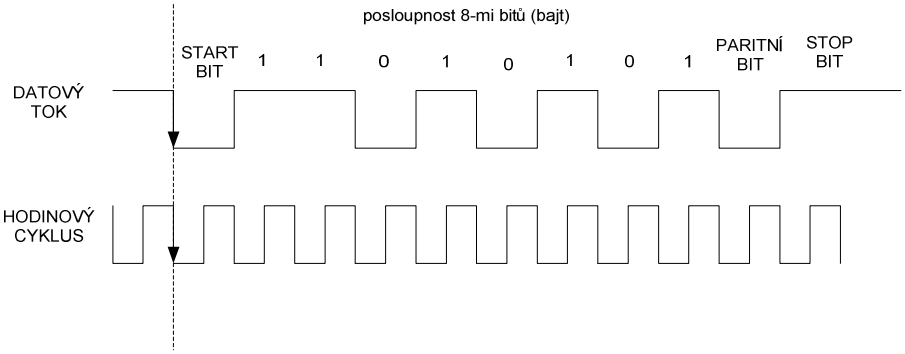
Takto vypadá část programu napsaná v assembleru pro nastavení registrů (asynchronní přenos) :

```
LDI    R16, 0B00000000
OUT    UCSR0A, R16              ; nastavení reg. UCSRA
LDI    R16, 0B00011000
OUT    UCSR0B, R16              ; nastavení reg. UCSRB
LDI    R16, 0B10000110
OUT    UCSR0C, R16              ; nastavení reg. UCSRC

LDI    R16, LO(11059200/ 16 / 9600 - 1)  ; nastavení reg. UBRR
OUT    UBRR0L, R16
LDI    R16, HI (11059200 / 16 / 9600 - 1)
OUT    UBRR0H, R16
```

Tímto nastavením povolíme příjem a vysílání (RXEN, TXEN), nastavíme přenosovou rychlost (U2X), počet datových bitů na 8 (UCSZ0,1,2), výběr registru UCSRC (7bit URSEL), zvolíme asynchronní komunikaci (UMSEL), zakážeme paritu (UPM0, 1), počet stopbitů jeden (USBS) a nastavíme přenosovou rychlost (UBRR).

3.2.5.2.1 Asynchronní přenos dat [17]

Přenos začíná start bitem, při kterém se logická hodnota na lince přepne z původního klidového stavu do opačného. Následuje n datových bitů (uspořádání bitů je od MSB – nejvýznamnějšího bitu po LSB – nejméně významný bit), paritní bit (pro kontrolu dat, může, ale nemusí být) a je ukončen jedním nebo více stop bity. Pro přenos máme k dispozici jednu datovou linku. Přijímač a vysílač má vlastní zdroj synchronizačního signálu. Každý přijímač se musí umět synchronizovat na start bit. K synchronizaci se používá *sestupná hrana start bitu*, za kterou následují odesílaná data. Start bit a stop bit mají rozdílnou hodnotu logické úrovně.

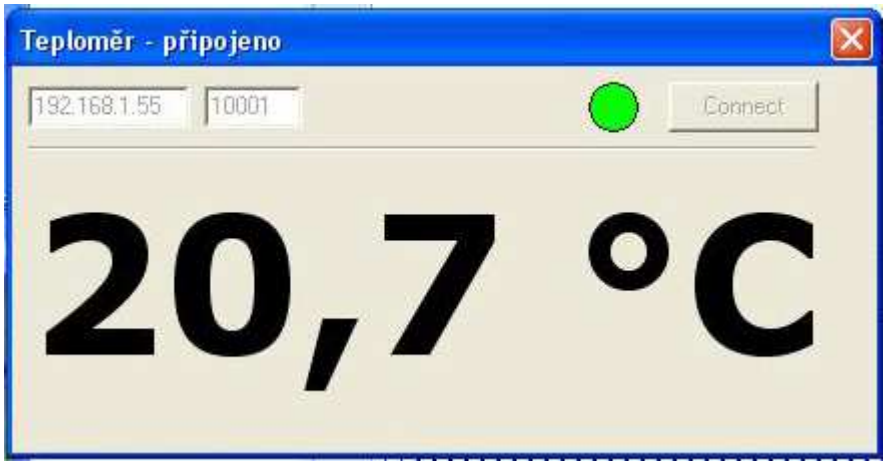


Obr. 10 Asynchronní přenos informací

3.2.6 Zpracování a zobrazení příchozích dat na straně příjemce (PC)

Pro komunikaci je nutné, aby modul XPort (*server*) a příjemce PC (*klient*) byly ve stejné síti (viz. kap. 3.3.10). Příchozí TCP/IP datagramy (viz kap. 3.3.7) musí být na straně příjemce zpracovány a vyhodnoceny.
Delphi (viz. kap. 4.2) disponuje komponentami, podporujícími komunikaci *klient x server* (viz. kap. 3.3.8) a umožňuje tak zpracovávat příchozí data. Pracuje se *sockety* (schránkami), které umožňují komunikaci mezi procesy (*klientem a serverem*). Proces na straně serveru řídí *socket* jako jeden konec komunikačního kanálu a proces na straně klienta využívá jiný *socket*, jak druhý konec kanálu. *Socket* je definován třemi atributy (doménou, typem a protokolem). Pro práci se sockety slouží knihovna *WinSock.* Delphi využívá vlastností OS, takže se o spoustu věcí nemusí starat. Dále využíváme vlastnosti *vláken (threads),* která se vykonávají paralelně s dalšími vlákny programu. Každý program má minimálně jedno vlákno (tzv. primární) *[20].*

Vytvořená aplikace se po spuštění inicializuje a jako *klient* se připojí k *serveru*, který dosud „naslouchal“ (připraven na připojení klienta). Pro připojení, musí *klient* kromě IP adresy uvést i číslo portu, na který se chce připojit (poslat žádost o spojení) (viz. Obr. 11). Aplikace zkontroluje hlavičku, kontrolní součty dat a v případě, vše proběhlo v pořádku, uloží data do bufferu. Data z bufferu jsou zpracována (vypočtena střída, která je následně přepočtena na teplotu) a zobrazena v příslušném formátu.



Obr. 11 Zobrazení naměřené teploty v Delphi

Po stisku tlačítka „Connect“ dojde k připojení, v opačném případě vyskočí varovná hláška. V levém horním rohu formuláře je zobrazen aktuální stav připojení. V případě že bude neměnná teplota (abychom věděli, že měření proběhlo) je v návrhu zakomponována „signálka“ (Shape), která změní barvu při každém měření ze zelené na červenou a ozve se zvukové znamení.

3.2.6.1 Přepočet střídy (DC) na teplotu (t)

Tento přepočet by bylo možné provést už v samotném mikroprocesoru a posílat do PC vypočítanou teplotu. Protože ale mikroprocesor nedisponuje instrukcemi v plovoucí desetinné čárce (*float*) a museli bychom implementovat aritmetiku pro typ float, je výhodnější provést přepočet až na straně příjemce. Budeme tedy přenášet pouze informaci o celkovém počtu vzorků a počtu vzorků s úrovní log. „1“. Z těchto údajů lze, s pomocí *Delphi* (popř. jiného vyššího programovacího jazyku) vypočítat jednoduše střídu (DC), převést střídu na teplotu a dále do správného formátu pro zobrazení. *Delphi* umí automaticky pracovat s 32bitovými opearandy.

Pro přepočet střídy (DC) na teplotu (t) použiji výrobcem snímače danou nominální rovnici DC = 0.32 + 0.0047 * t, ze které si vyjádřím teplotu následujícím postupem :

$$\mathrm{DC} = 0{,}32 + 0{,}0047 \cdot \mathrm{t}, \tag{3.4}$$

$$\mathrm{t} = \frac{\mathrm{DC} - 0{,}32}{0{,}0047}. \tag{3.5}$$

Výpočet střídy a její přepočet na teplotu napsaný v Delphi vypadá následovně :

```
var                          // deklarace proměnných
  a: array[0..1] of integer;
  t : real;

begin
  if Len <> SizeOf(a) then
    Exit;

  Move(Data^, a, SizeOf(a));     // přesune data z bufferu do proměnné „a“

  t : = ((a[0] / a[1]) - 0.32) / 0.0047;      // přepočte střídu DC (a[0] /
a[1) na teplotu

  Label1.Caption := Format(SLabel, [t]);      // zobrazení
                                // SLabel je definovaný jako SLabel = ' %.1f
°C ‘
end;
```

Pozn. V proměnné **a[0]** je počet log. „1“ (TXDATA) a v **a[1]** je celkový počet snímaných vzorků v dané periodě (PCT).

3.3 XPort XE – převodník rozhraní *[4]*

3.3.1 Základní popis modulu

XPort XE je kompletní převodník rozhranní Ethernet na sériovou linku RS232 s úrovněmi TTL. Tento modul nám umožňuje síťovou komunikaci pro naše zařízení. Na trhu je možnost výběru z více variant (XPort XE, XPort SE, XPort 485). Pro požadované funkce postačí varianta XPort XE s úrovněmi TTL.

3.3.2 Složení převodníku

Modul je vložen do krytu konektoru RJ-45, uvnitř kterého se nachází procesor DSTni-LX 186 s řadiči, paměť, obvody rozhraní Ethernet 10/100, rychlý port RS232 ,diagnostické LED a 3 programovatelné I/O piny. Pro připojení k LAN/WAN síti nebo internetu, obsahuje XPort kompletní TCP/IP sadu a operační systém. Další součástí je výkonný terminálový a WWW server, který nabízí kompletní rozhraní 10Base-T/100Base-TX Ethernet (Fast Ethernet s připojením kroucenou dvoulinkou) a lze jej využít například pro dálkovou konfiguraci, dohled, vizualizaci a ovládaní připojeného zařízení.
Firmware modulu XPort pracuje na čipové sadě s procesorem DSTni-LX vybavené 256KB SRAM, 2KB ROM zavaděče a MAC s integrovaným 10/100BaseTX PHY. XPort může komunikovat se zařízením prostřednictvím sériového rozhraní v úrovních 3.3V a dalších 3 programovatelných vstupů/výstupů. Pro firmware a www stránky je k dispozici 512KB paměti flash. XPort je napájen napětím 3.3V a obsahuje vestavěný obvod dohledu napájecího napětí, který aktivuje reset v případě, že napětí poklesne pod úroveň 3.08V zajišťující spolehlivý provoz.
Vstavěný stabilizátor 2.5V zajišťuje napájení jádra procesoru.
Xport je navržen tak aby splňoval podmínky požadavků na EMC dle
EN 55022 - vyzařovaní úrovně B.

3.3.3 Vlastnosti modulu

- převod s sériového rozhraní na Ethernet
- vestavěný WWW server
- kompletní řešení inteligentního modulu v konektoru RJ-45
- implementace TCP/IP a SW
- prostředí pro aplikace založené na technologii WWW
- 10/100Base-T Ethernet
- snadná konfigurace pomocí WEB rozhraní
- interaktivní WEB stránky s použitím Java Appletů
- režie spojená s komunikací se sítí je zcela pokryta prostředky XPortu
- upgrade firmware přes síť
- shoda s požadavky na EMC
- velký dostupný výpočetní výkon a komunikační propustnost -12 MIPS při 48 MHz, 10/100M Ethernet
- rozšířený rozsah pracovních teplot -40 to +85°C

3.3.4 Protokoly využívané XPortem

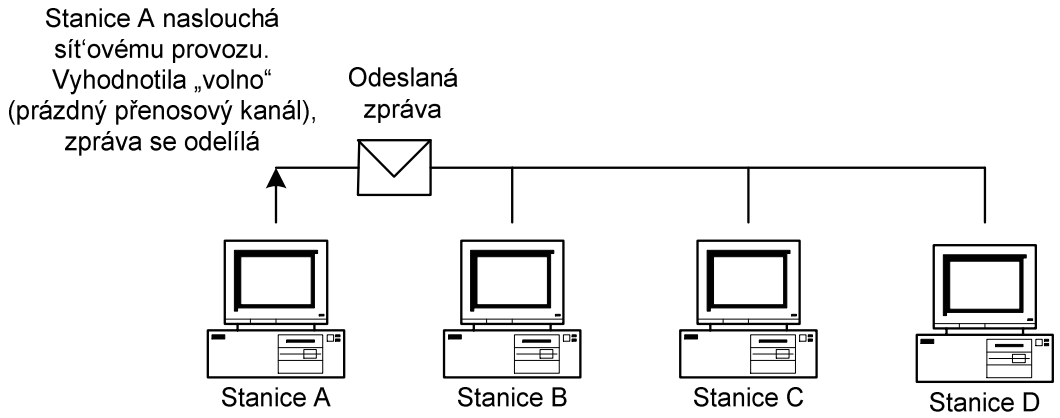
Pro síťovou komunikaci používá modul protokol IP (Internet Protocol) a pro zajištění ochrany proti ztrátě dat, duplikaci a zachování pořadí posílaných dat protokol TCP (Transmission Control Protocol). Ostatní podporované protokoly jsou : ARP, UDP, TCP, ICMP, Telnet, TFTP, DHCP, HTTP, SNMP, TCP, UDP, TFTP, IP, UDP.

3.3.5 Sériové rozhraní RS 232 *[17]*

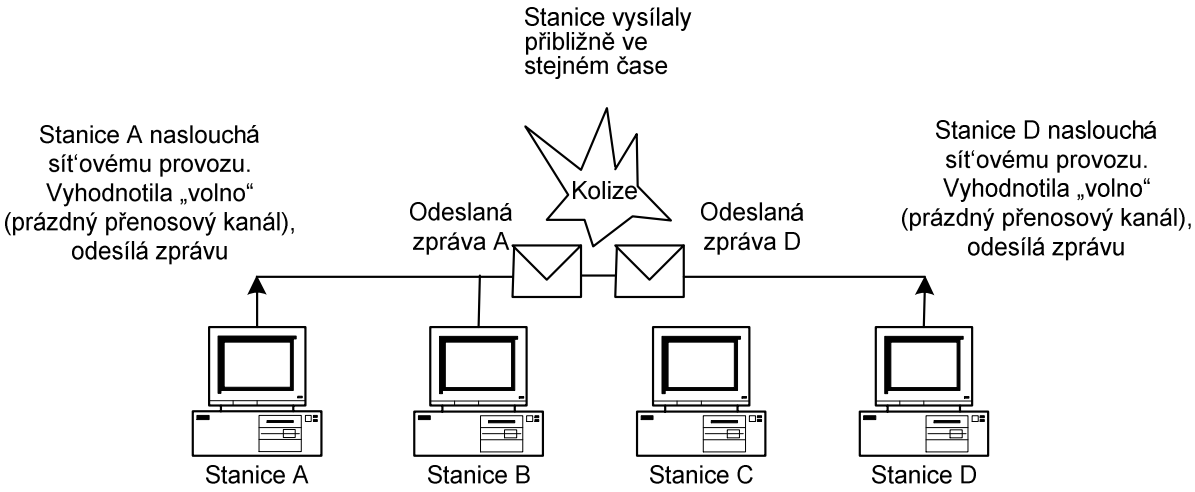
Sériové rozhraní umožňuje vzájemnou sériovou komunikaci dvou zařízení. Standard určuje, jak přenést jednotlivé bity po vodiči. Bity každého znaku se přenáší postupně za sebou. Viz. asynchronní přenos dat, který je popsán v kapitole 3.2.5.2.1.

3.3.6 Ethernet *[18]*

Síť typu Ethernet se dnes téměř stala standardem. Využívá tzv. sběrnicovou topologii, kdy médium v síti Ethernet je společné pro všechna zařízení připojené do daného segmentu sítě LAN. Všechna zařízení před vysíláním naslouchají síťovému provozu (monitorují síť) a k vyslání své vlastní zprávy dojde až v případě, kdy je médium (např. síťový kabel) prázdný. Dojde-li k vysílaní dvou stanic přibližně ve stejném čase, jejich pakety se střetnou (dojde ke kolizi) a obě vysílání zruší. Počkají náhodný časový interval po kterém dojde k opětovnému vysílání. Pro přístup ke společnému přenosovému médiu je využita metoda **CSMA/CD** (Carrier Sence with Multiple Access and Collision Detection). Tato metoda řídí naslouchání síťového provozu, detekci kolizí, přerušení, atd. Po úspěšném odeslání zprávy uzlem do sítě obdrží vysílanou zprávu všechny stanice. Každá stanice vyhodnotí, je-li cílová adresa zprávy stejná jako jeho vlastní. Pokud se shodují, zařízení zprávu přijme a zpracuje. Pokud ne, zprávu zahodí. Každé zařízení v síti je identifikováno svou vlastní hardwarovou adresou (MAC adress).



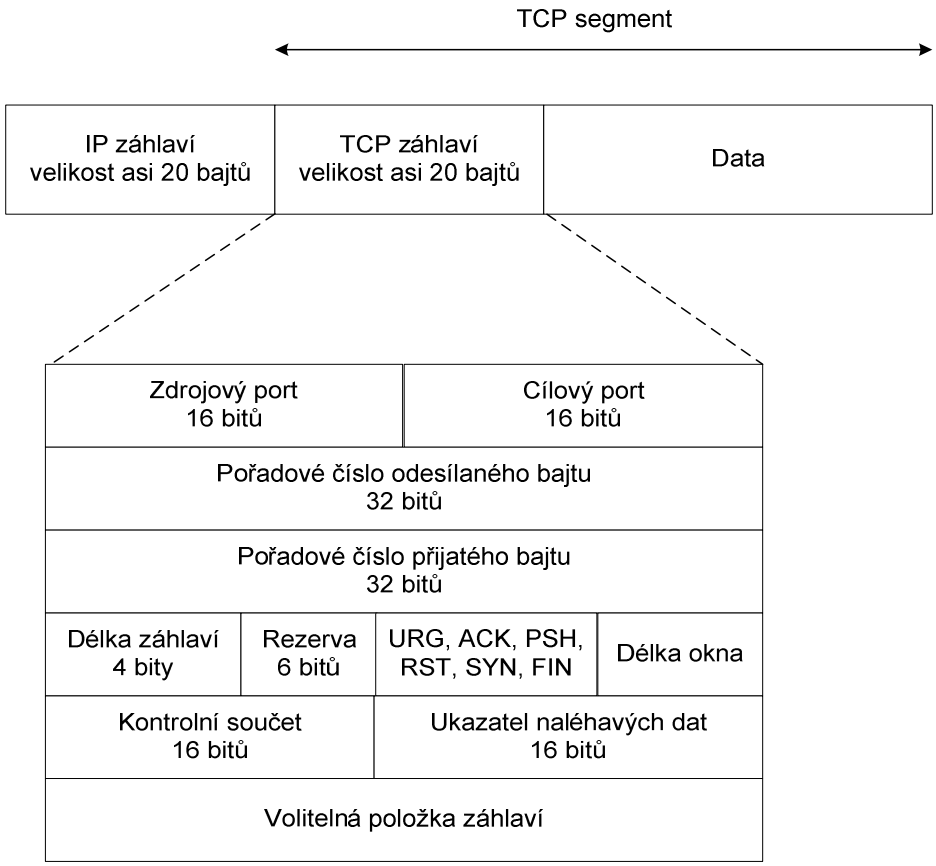
Obr. 12 Stanice naslouchá a zahájí vysílání



Obr. 13 Detekce kolize

3.3.7 Protokol TCP/IP *[17]*

Síť internet pracuje nad protokoly TCP/IP (Transmission Control Protocol/Internet Protocol). Zkratka TCP/IP označuje ve skutečnosti celý balík protokolů, z nichž každý hraje svou určitou roli.
Protokol TCP (transportní vrstva) je narozdíl od IP (síťová vrstva), protokolem vyšší vrstvy. TCP dopravuje data mezi dvěma konkrétními aplikacemi, zatímco IP mezi libovolnými počítači v internetu. Protokol TCP využívá k transportu dat internetem protokol IP, avšak nad tímto protokolem zřizuje spojovanou službu. Tj. že mezi dvěma aplikacemi naváže spojení. Vytvoří na dobu spojení virtuální okruh, který je plně duplexní - data se přenášejí současně na sobě nezávisle oběma směry. Musí řešit problémy navázání a ukončení spojení, potvrzování přijatých dat (přenášené bajty jsou číslovány, integrita přenášených dat je zabezpečena kontrolním součtem), vyžádání ztracených dat (ztracená nebo poškozená data jsou znovu vyžádána), ale také problémy průchodnosti přenosové cesty. Aplikace používající protokol TCP si nemusí dělat starosti s tím, zdali náhodou nebyla nějaká data během přenosu ztracena nebo díky chybě přenosu pozměněna.
Cílová aplikace je v internetu jednoznačně určena IP-adresou, číslem portu a použitým protokolem (TCP nebo UDP). Protokol IP dopraví IP-datagram na konkrétní počítač, kde běží jednotlivé aplikace a podle čísla cílového portu operační systém pozná, které aplikaci má TCP segment doručit. Základní přenosovou jednotkou protokolu TCP je TCP segment. TCP Segment se vkládá do IP-datagramu. IP-datagram zase do linkového rámce.



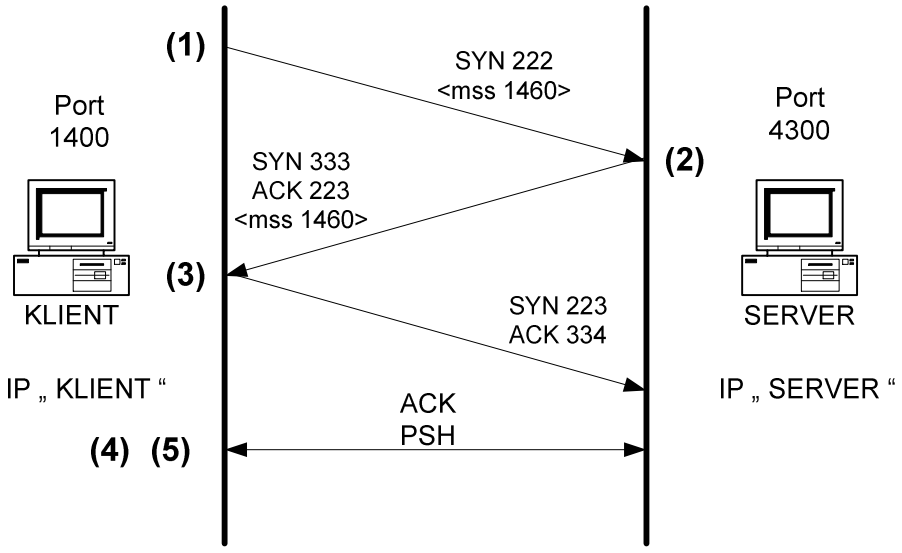
Obr. 14 TCP segment

3.3.8 Komunikace protokolem TCP (Klient x Server) *[17]*

Komunikace popisuje výměnu segmentu TCP mezi oběma konci TCP spojení (tzv. handshaking). Stranou navazující spojení bude tzv.klient (na portu např. 1400) a strana, která spojení očekává tzv.server (port 4300). Port serveru musí být předem všeobecně znám, zatímco port klienta se může měnit (požádá o dočasné přidělení volného portu, operační systém přiděluje porty 1023 a větší – tzv. klientské, náš server s portem 4300 tak může spustit kterýkoli běžný uživatel).

3.3.8.1 Navázání spojení

Klient začne navazovat spojení odesláním prvního TCP segmentu na port serveru (4300), který vloží do IP-datagramu, jehož IP adresa odesílatele bude „klient“ a příjemce „server“. Klient vygeneruje náhodné číslo v intervalu od 0 do 2^{32} - 1, které použije jako startovací číslo odesílaného bajtu (tzv. ISN). Zvolíme pro znázornění např. ISN = 222. Vytvořením startovacího čísla bajtu se nastaví v segmentu příznak SYN. Segment je prvním segmentem v TCP komunikaci, proto nemůže potvrzovat žádná přijatá data. Pole pořadové číslo přijatého bajtu nemá platný význam (bývá vyplněno binárními nulami) a nemůže být tedy ani nastaven příznak ACK (příznak ACK je nastaven ve všech dalších TCP segmentech až do ukončení spojení).
Součástí segmentů **„1“** a **„2“** byla volitelná položka záhlaví TCP segmentu MSS (*Maximum segment size*). Tato položka oznamuje druhé straně maximální délku datové části TCP segmentu jakou si přeje přijímat. Hodnota MSS je pro použití mimo lokální síť 536 bajtů (WAN) a pro linkový protokol Ethernet II 1460 bajtů. Tato volba se může vyskytovat v TCP segmentech s nastaveným příznakem SYN (v prvních dvou segmentech).
Třetí segment **„3“** rovněž potvrzuje příznak SYN tak, že jakoby potvrzuje jeden bajt. Třetí a další segment již nemůže nést volitelnou položku záhlaví MSS (maximální délka segmentu). Třetím segmentem končí navazování spojení.
Čtvrtý segment **„4“** posílá klient serveru. Jakmile kterákoliv strana obdrží první ACK, tak může začít vysílat (TCP je plně duplexní spoj). Fakt, že TCP segment nese aplikační data je vyjádřena nastavením příznaku PSH.
V případě, že jeden konec spojení odesílá data a druhý nemá momentálně žádná data k odeslání, tak i přesto musí potvrzovat přijatá data. Potvrzování provádí TCP segmenty s nenastaveným příznakem PSH (segmenty bez dat), ale s příkazem ACK.



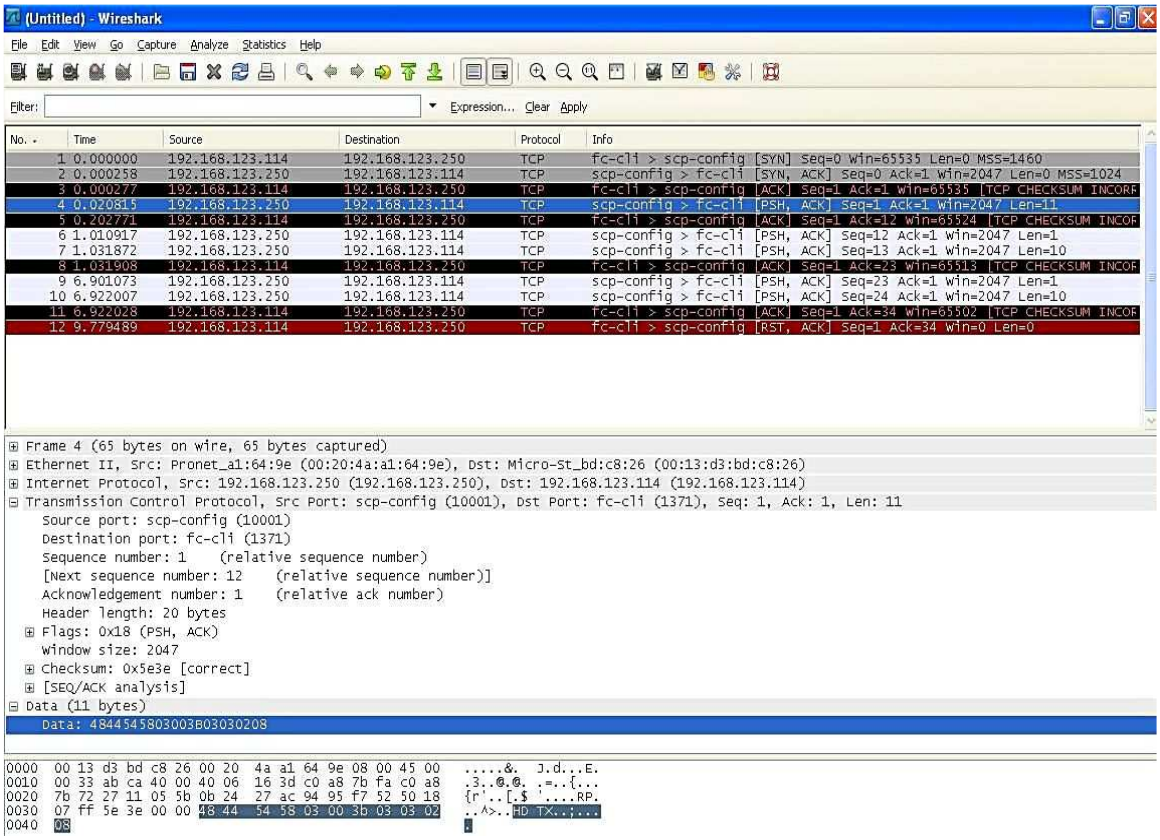
Obr. 15 Příklad navázání spojení klient x server

3.3.8.2 Ukončení spojení

Ukončení spojení může generovat kterákoli strana odesláním TCP segmentu s příznakem FIN, tzv. aktivním ukončením spojení (nemůže odeslat TCP segment s příkazem PSH). Druhá strana reaguje tzv. pasivním ukončením spojení (může v odesílání dat pokračovat a to do doby,než sama spojení ukončí). Pro patřičné uzavření spojení jsou nutné 4 TCP segmenty. Příkaz FIN se stejně jako SYN potvrzuje.

3.3.9 Zachycení komunikace protokolu TCP (Klient x Server)

Pro ověření komunikace typu klient x server popsané v kapitole 3.3.8 jsem použil volně stažitelný program Wireshark, který se využívá pro analýzu síťového provozu. Zachytává veškerou komunikaci jdoucí skrze síťové rozhraní našeho počítače. Umožňuje použití nejrůznějších filtrů.



Obr. 16 Komunikace Klient x Server zachycená Wiresharkem

Na Obr.16 jsou znázorněny IP-datagramy, zachycené při komunikaci Klent x Server. Z*drojová* (192.168.123.114) a *cílová* (192.168.123.250) *IP adresa*, *zdrojový* (10001) a *cílový* (1371) *port*, příznaky a především přenášená hlavička protokolu „HDTX“, kterou jsme nadefinovali. Jako zdroj (klient) zde vystupuje PC (rsp. jeho síťová karta) a jako cíl (server) modul XPort.

V tab. 4 je znázorněno navázání a ukončení spojení. Nastavení jednotlivých příznaků v každém segmentu.

Tab. 4 Příznaky při komunikaci zachycené Wiresharkem

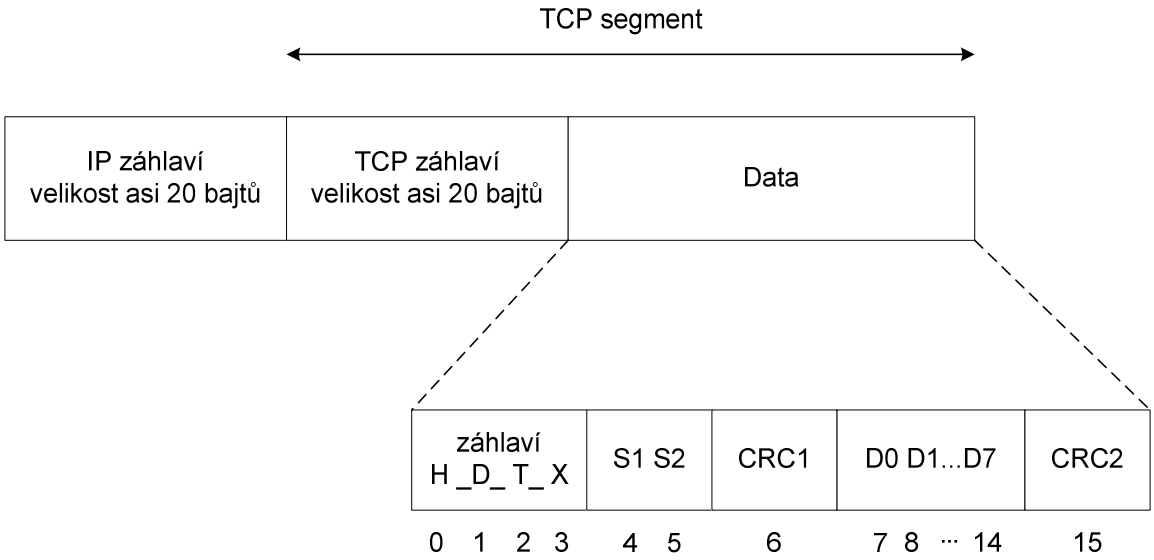
Time (s)	192.168.123.114	192.168.123.250	
0,000	SYN		Seq = 0 Ack = 4108048469
	(1371) ---------->	(10001)	
0,000	SYN, ACK		Seq = 0 Ack = 1
	(1371) <----------	(10001)	
0,000	ACK		Seq = 1 Ack = 1
	(1371) ---------->	(10001)	
0,021	PSH, ACK - Len: 11		Seq = 1 Ack = 1
	(1371) <----------	(10001)	
0,203	ACK		Seq = 1 Ack = 12
	(1371) ---------->	(10001)	
1,011	PSH, ACK - Len: 1		Seq = 12 Ack = 1
	(1371) <----------	(10001)	
1,032	PSH, ACK - Len: 10		Seq = 13 Ack = 1
	(1371) <----------	(10001)	
1,032	ACK		Seq = 1 Ack = 23
	(1371) ---------->	(10001)	
6,901	PSH, ACK - Len: 1		Seq = 23 Ack = 1
	(1371) <----------	(10001)	
6,922	PSH, ACK - Len: 10		Seq = 24 Ack = 1
	(1371) <----------	(10001)	
6,922	ACK		Seq = 1 Ack = 34
	(1371) ---------->	(10001)	
9,779	RST, ACK		Seq = 1 Ack = 34

3.3.10 Nastavení XPortu pro síťovou komunikaci *[5]*

Modul XPort je vybaven interním WEB serverem, který umožňuje pohodlné nastavení a ovládání modulu. V základu má modul přednastavenou počáteční IP adresu. MAC adresu má pevně přidělenou výrobcem. Pro připojení na WEB server modulu lze využít např. WEB prohlížeč (Internet Explorer, Mozilla...), kam je možno zadat patřičnou IP adresu a připojit se. Před připojením je však nutné, aby XPort a PC (síťová karta) byli ve stejné síti. Pro přímé „fyzické“ propojení modulu a PC lze využít křížený kabel (cross kabel). Při připojení přes swich, hub, router, atd. je možno využít přímý kabel.
Pro sériovou komunikaci musí být na XPortu v záložce „Port Settings“ nastaven **protokol RS232**, **Baud Rate** na **9600** a **Data Bits** na **8.**
Pro komunikaci (klient x server) v záložce „Network“ a „Connection“ **číslo lokálního portu**, **IP adresu** na kterou se chceme připojit a **použitý komunikační protokol.**

3.4 Vlastní protokol

Komunikace se bude odehrávat pomocí TCP/IP datagramů přenášených skrze síť. Není tedy třeba řešit problémy s navázáním a ukončením spojení, potvrzením přijatých dat, vyžádáním ztracených dat, ani problémy s průchodností přenosové cesty. O všechno se postará protokol TCP. Pro naše účely vytvořený protokol bude pracovat nad úrovní TCP.



Obr. 17 Blokové znázornění našeho protokolu

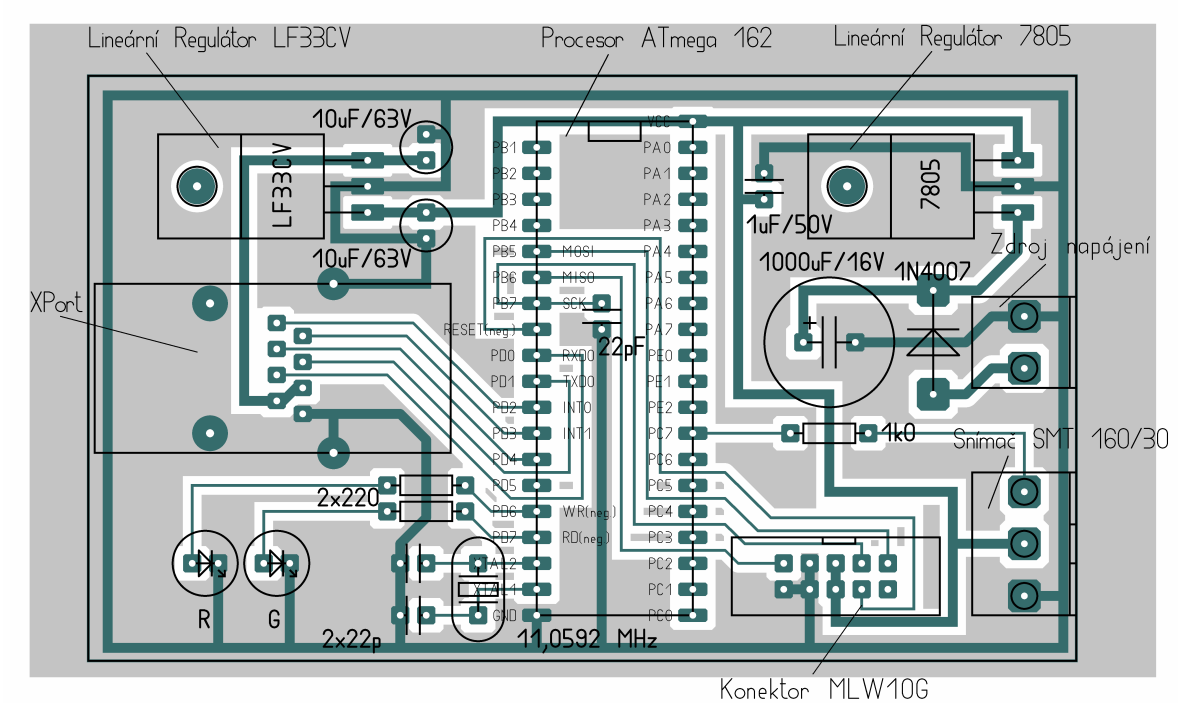
- záhlaví H_D_T_X - přenášené znaky (4 bajty)
- S1 S2 - velikost dat (2 bajty)
- CRC1 - kontrolní součet dat a hlavičky (1 bajt)
- D0 D1... D7 - přenášená data (8 bajtů)
- CRC2 - kontrolní součet dat (1 bajt)

Část zdrojového kódu pro vytvoření protokolu napsaná v assembleru :

```
;tvorba      hlavičky "HDTX"

             LDI    R16, TXSIZE - TXHEAD      ; identifikační string (R16 = 4)

             LDI    YL, LO(TXHEAD)            ; do YL odkaz na adresu TXHEAD
             LDI    YH, HI(TXHEAD)

             LDI    ZL, LO(HEADST * 2)        ; do ZL spodní adresa HEADST
             LDI    ZH, HI(HEADST * 2)        ; do ZH horní adresa HEADST

LOOP1:       LPM    R18, Z+                   ; přečte z adresy (flash) Z a
                                              ; zapíše do R18, inkrementuje Z
             ST     Y+, R18                   ; z reg.R18 zapíše hodnotu
                                              ; znaku na adresu Y a
                                              ; inkrementuje Y

             SUBI   R16, 1                    ; odečte z reg.R16 hodnotu „1“
             BRNE   LOOP1                     ; bude-li hodnota předchozí
                                              ; instrukce = 0, tak pokračuj...

;vypočet kontrolní velikosti dat

             LDI    R16, LO(TXCRC2 - TXDATA)  ; identifikační string
             LDI    R17, HI(TXCRC2 - TXDATA)

             STS    TXSIZE + 0, R16           ; uložení 16-ti bitových dat
             STS    TXSIZE + 1, R17

;vypočítat CRC hlavičky

             EOR    R18, R18                  ; vynuluje R18

             LDI    R16, TXCRC1 - TXHEAD      ; uložení 8-mi bitových dat
             LDI    ZL, LO(TXHEAD)            ; nastavení ukazatele
             LDI    ZH, HI(TXHEAD)

LOOP2: LD    R19, Z+                           ; načte data z pozice
                                              ; ukazatele Z do reg.19
             ADD    R18, R19                  ; součet dat, který se uloží do
R18
             SUBI   R16, 1                    ; odečítá od reg.R16 hodnotu "1"
             BRNE   LOOP2

             STS    TXCRC1, R18               ; uložení přímo adresovaných dat

;vypočítat CRC dat

             EOR    R18, R18

             LDI    R16, TXCRC2 - TXDATA
             LDI    ZL, LO(TXDATA)            ; ukazatel na data v paměti
             LDI    ZH, HI(TXDATA)
      LOOP3: LD     R19, Z+                   ; data z paměti zkopíruje
                                              ; do registru
             ADD    R18, R19                  ; součet dat v R18

             SUBI   R16, 1                    ; odečítá od reg.R16 hodnotu "1“
             BRNE   LOOP3

             STS    TXCRC2, R18               ; zápis kontrolního součtu
```

3.5 Návrh desky plošných spojů

3.5.1 Osazená deska plošných spojů

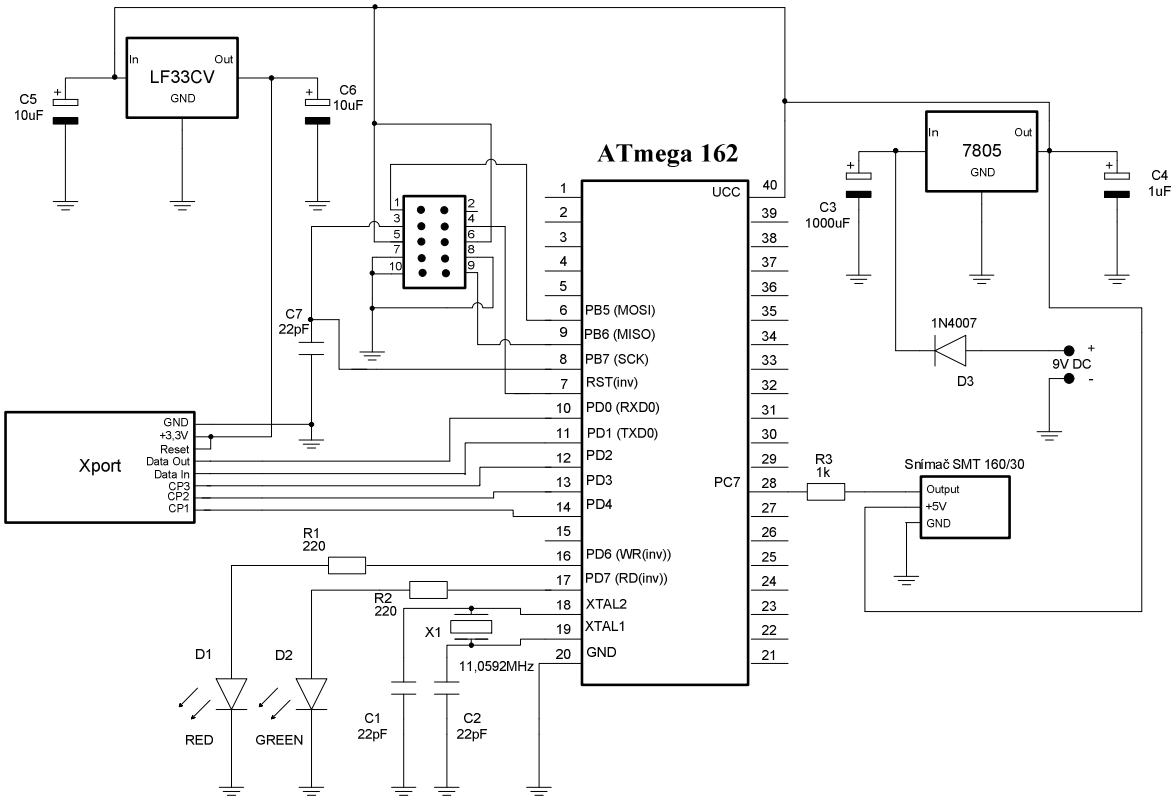


Obr. 18 Návrh desky plošných spojů

Na Obr. 18 je návrh plošné desky spojů. Při návrhu zapojení jsem vycházel z datasheetů jednotlivých komponent popsaných v předchozích kapitolách. Jedná se o následující komponenty :

- **lineární regulátor LF33CV** (pro stabilizaci napájení Xportu 3,3V) *[7]*,
- **lineární regulátor 7805** (pro stabilizaci napájení Procesoru a programátoru 5,0V) *[8]* ,
- **procesor ATmega 162/30** *[3]* **,**
- **XPort XE** *[4]* ,
- **krystal** (11,0595MHz) *[14]*,
- **snímač SMT 160/30** *[2]* ,
- **konektor MLW10G** (pro připojení programátoru) *[15]* ,
- **elektrolytický kondenzátor 2x** (10µF/63V) ,
- **keramický kondenzátor 3x** (22pF) *[13]* ,
- **elektrolytický kondenzátor 1x** (1µF/50V) *[12]* ,
- **radiální elektrolytický kondenzátor 1x** (1000µF/16V) *[11]* ,
- **usměrňovací dioda 1N4007** (1000V, 1A) *[10]*,
- **odpor 3x** (2x220Ω, 1x1k Ω) ,
- **LED diody 2x** (5mm, 90mCd / 20mA, úhel 30°) *[9]* .

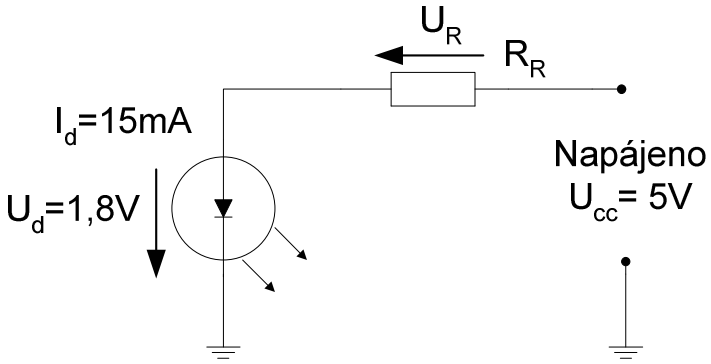
3.5.2 Popis činnosti zařízení



Obr. 19 Schéma zapojení plošné desky

Napájení zařízení na desce plošných spojů je provedeno vnějším zdrojem stejnosměrného napájení AC Adaptor (model AD 0900350DV) s výstupním napětím 9V/350mA DC. Napájecí napětí je vedeno přes ochranou diodu D3 na stabilizátor napětí 7805, kterým je upraveno napětí na stabilizovaných +5V pro napájení snímače SMT160/30, procesoru ATmega162 a konektoru programátoru.
Pomocí dalšího stabilizátoru je vytvořeno napájecí napětí +3,3V pro převodník rozhraní Xport.
Pro zapojení regulátorů LF33CV a 7805, procesoru ATmega162, snímače SMT160/30, převodníku rozhraní Xport a konektoru programátoru MLW10G jsem použil doporučených schémat z datasheetů. Kromě datasheetů a zde uvedených doporučených zapojení jsem zařadil do vstupního obvodu procesoru ze strany snímače ochranný odpor o hodnotě 1kΩ/125mW, který chrání procesor v případě zkratu na vstupním čidle SMT 160/30 . Výkonové zatížení ochranného odporu při zkratu je P = U2 / R = 0,025W = 25 mW.
Mezi vstupy XTAL1 a XTAL2 procesoru ATmega162/30 je zapojen krystal, který je uzemněn pomocí dvou keramických kondenzátorů s hodnotou 22pF [7].

Pro signalizační LED diody jsem použil omezovací odpory 220Ω/250mW , jejichž hodnotu jsem určil z následujícího zapojení (viz. Obr. 20) :



Obr. 20 Zapojení LED diody

$$R_R = \frac{U_{CC} - U_d}{I_d} \ , \qquad (3.6)$$

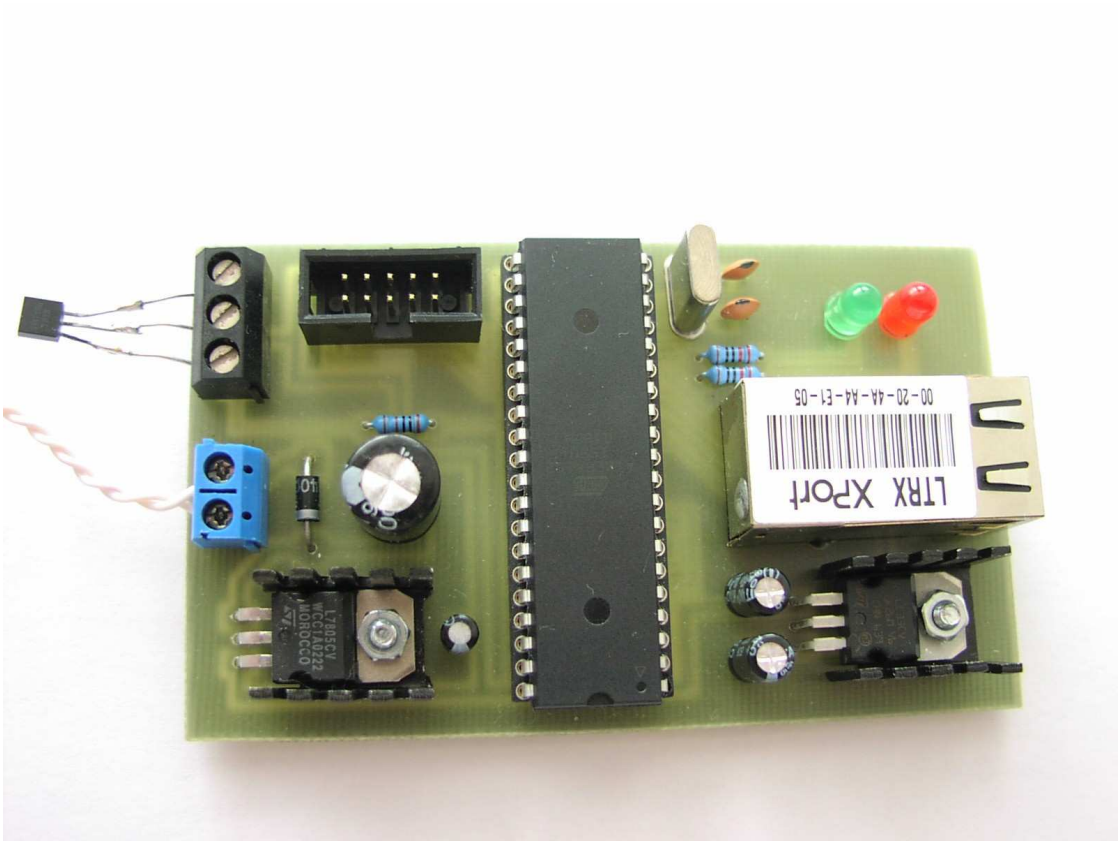
$$R_R = \frac{5-1{,}8}{0{,}15} = \underline{210\Omega} \ , \qquad (3.7)$$

$$P_R = \frac{U_R{}^2}{R_R} \ , \qquad (3.8)$$

$$P_R = \frac{3{,}2}{220} = 0{,}047\mathrm{W} = \underline{47\mathrm{mW}} \ . \qquad (3.9)$$

Z řady volíme odpor $\underline{R_R = 220\Omega}$. Provozní výkonové zatížení P_R = 47mW. Při realizaci byly použity odpory s výkonovým zatížením 250mW z důvodů odolnosti proti meznímu zatížení při zkratu LED diody (plné napájecí napětí na odporu).
Signál ze snímače je zpracován procesorem (viz. kap. 3.2.5). Výsledek zpracování je vysílán přes rozhraní RS232 na Xport, který převede rozhraní RS232 na Ethernet. Data jsou pomocí TCP/IP datagramů přenášena skrze síť na PC, kde jsou zpracována (viz. kap. 3.2.6). Měřená hodnota teploty je zobrazena na monitoru PC.

3.6 Foto desky funkčního vzorku



Obr. 21 Foto desky funkčního vzorku

4 Software

4.1 Software k mikroprocesoru

K programování procesoru jsem použil integrované vývojové prostředí ***WinAVR***, které je komplexně založeno na řadě mikroprocesorů ATMEL AVR. Tento program disponuje všemi důležitými a potřebnými prvky, počínaje editorem kódu (s barevným zvýrazněním symbolů), přes překladač a debugger (analýza programu, umožňující krokování a trasování, zobrazení I/O registrů a obsahů všech pamětí), až po nastavování a definování obsahů I/O registrů. Editor vstupních průběhů umožňuje na portech definovat vstupní průběhy a záznam průběhů následně zobrazí výstupní stavy v libovolném časovém měřítku. Tyto funkce jsou ideální pro ladění a ukázku sériového kanálu popřípadě jiné komunikace s PC. V tomto programu lze i přímo přes rozhraní ISP programovat všechny typy mikroprocesorů ATMEL AVR.

4.2 Software na straně přijímače (PC) *[20]*

Program na straně přijímače musí umět zpracovat příchozí data a zobrazit je ve vhodném grafickém prostředí. Aby tvorba aplikace nebyla zbytečně zdlouhavá a náročná, je třeba použít vhodný software.
Při programování ve vývojovém nástroji ***Delphi*** má programátor maximálně zjednodušenou návrhovou fázi. Vytvoření vizuálního prostředí zvládne skoro každý. Mezi hlavní přednosti tohoto vývojového nástroje patří :

- firma *Borland*, která má *Delphi* vyvinula a udržuje je známým výrobcem spolehlivých, osvědčených překladačů pro programovací jazyky Pascal a C ,
- jednoduchý návrh vizuálního prostředí aplikace s využitím vlastností operačního systému (o mnohé věci se nemusíme starat),
- objektově orientovaný přístup ,
- množství integrovaných nástrojů ,
- významná podpora Internetu ,
- možnost vytvářet aplikace ***klient/server (knihovna WinSock) ,***
- velké množství volně dostupných (již hotových) komponent a možnost vytvářet komponenty vlastní .

Delphi je určeno pro operační systémy *Windows.* První verze *Delphi* byli určeny pro 16-ti bitové prostředí, od verze 2 je překladač plně 32-bitový.

4.3 Software pro návrh desky plošných spojů *[16]*

Pro návrh desky jsem použil program PlotPC, se kterým jsem se už v minulosti setkal. Obsahuje funkce, které umožňují snadné vytváření spojového obrazce. Základní funkce programu lze zvládnout prakticky okamžitě a bez návodu.
Deska plošných spojů je v tomto grafickém editoru vytvořena pomocí soustavy čar (spojů), pájecích plošek (bodů), obdélníkového nebo kruhového tvaru a pomocných tvarů např. oblouků (tzv. objektů).
Na desce je možno současně použít 16 typů čar různé tloušťky, 16 velikostí bodů pravoúhlých a 16 velikostí bodů kruhových. Typy jsou jednoduše značeny písmeny A až P. Lze tedy mluvit o bodu typu A, čáře typu C apod. Na obrazovce jsou typy kromě velikosti odlišeny také barvou. Přiřazení určité barvy k určitému typu je si volí uživatel při kreslení desky. Stejně tak skutečné velikosti jednotlivých typů.

5 Závěr

Cílem závěrečné práce bylo navrhnout dálkovou komunikaci prostřednictvím rozhraní Ethernet mezi informačním zařízením a PC a tuto komunikaci prakticky realizovat. Po dohodě s vedoucím práce jsem realizoval dálkové měření teploty.

Práce obsahuje návrh hardwaru včetně zapojení a realizace desky plošných spojů. Dále návrh a realizaci programového vybavení zabezpečujícího přenos a vyhodnocení dat. Základem měřícího modulu je procesor ATmega162. Tento modul dálkově komunikuje s rozhranním Ethernet příslušného PC. Teplota je měřena nepřetržitě a zobrazována na monitoru PC. Podmínkou navázání komunikace je připojení PC do stejné sítě s měřícím modulem.

Vzhledem k rozsahu řešené problematiky bylo zadání po dohodě s vedoucím práce drobně modifikováno a rozšířeno o realizaci funkčního vzorku. Správnost návrhu i realizace projektu byla prakticky ověřena srovnáním dálkově naměřené teploty s údajem referenčního teploměru. Byla potvrzena i správnost zvolený přenosového protokolu, který dokáže účinně detekovat a eliminovat přenosové chyby a tím zabezpečuje spolehlivost přenosu.

Prameny

[1] Hw_group. http://www.hw-group.com/products/charon1/index_cz.html

[2] Omnitron s.r.o (snímač SMT 160/30 datasheet). http://www.omnitron.cz/

[3] Atmel corporation (procesor ATmega162 a příslušný datasheet). http://www.atmel.com/dyn/resources/prod_documents/doc2513.pdf

[4] Lantronix (XPort XE příslušný datasheet). http://www.lantronix.com/pdf/XPort_DS.pdf (Xport)

[5] Papouch s.r.o (stránky výrobce XPort) http://www.papouch.com/shop/scripts/_detail.asp?katcislo=0030

[6] Programátor. http://winide51.wz.cz/avr/prog.php

[7] Datasheet lineárního regulátoru LF33CV. http://www.datasheetcatalog.com/datasheets_pdf/L/F/3/3/LF33CV5V.shtml

[8] Datasheet lineárního regulátoru 7805. http://www.datasheetcatalog.com/datasheets_pdf/7/8/0/5/7805.shtml

[9] Datasheet LED diody. http://www.flajzar.cz/ke_stazeni/LR05D.pdf

[10] Datasheet usměrňovací diody 1N4007. http://www.datasheetcatalog.com/datasheets_pdf/1/N/4/0/1N4007.shtml

[11] Datasheet elektrolytického kondenzátoru E1000M/16V. http://www.gme.cz/_dokumentace/dokumenty/123/123-209/dsh.123-209.1.pdf

[12] Datasheet elektrolytického kondenzátoru1µF/50V. http://products.nichicon.co.jp/en/pdf/XJA043/e-st.pdf

[13] Datasheet keramického kondenzátoru 22pF. http://www.gme.cz/_dokumentace/dokumenty/120/120-262/dsh.120-262.1.pdf

[14] Datasheet krystalu. http://www.ges.cz/sheets/h/hc49u.pdf

[15] Datasheet konektoru MLW10G. http://www.gme.cz/_dokumentace/dokumenty/800/800-035/dsh.800-035.1.pdf

[16] Editor plošných spojů. http://sosak.xf.cz/download/Plot_pc.rar

Seznam literatury

[17] Dostálek, Libor a Kabelová, Alena. **Velký průvodce protokoly TCP/IP a systémem DSN**. Praha: Computer Press, 2000.

[18] Velte, T. Anthony a Velte, J. Toby. **Síťové technologie Cisto - velký průvodce.** Brno: Computer Press, 2003.

[19] Matoušek, David. **Práce s mikrokontroléry ATMEL AVR**. Praha: nakladatelství BEN – technická literatura, 2003.

[20] Kadlec, Václav. **Učíme se programovat v Delphi a jazyce Object Pascal**. Praha: Computer Press, 2001.

Seznam použitých zkratek,veličin a symbolů

OS	operační systém - Operating System
PC	osobní počítač - Personal Computer
SW	programová část - Software
HW	fyzická část - Hardware
ISP	programování v systému - In System Programming
SPI	sériové programovací rozhraní - Serial Interface Programming
ALU	aritmeticko-logická jednotka - Aritmetic Logic Unit
PC	čítač programu - Program Counter
ID	dekodér instrukcí - Instruction Decoder
SP	ukazatel vrcholu zásobníku - Stack Pointer
MISO	hlavní datový výstup
MOSI	hlavní datový vstup
RST	reset
SCK, XCK	generátor hodinových pulsů
LSB	nejméně významný bit - Least Significant Bit
MSB	nejvíce významný bit - Most Significant Bit
LAN	lokální počítačová síť - Local Area Network
WAN	dálková počítačová síť - Wide Area Network
TCP	Transmission Control Protocol
IP	Internet Protocol
EMC	elektromagnetická kompatibilita - Electromagnetic Compatibility
ROM	elektronická paměť - Read Only Memory
RAM	elektronická paměť - Random Access Memory
CMSA/CD	Carrier Sence with Multiple Access and Collision Detection
MMS	maximální velikost segmentu - maximum segment size
DC	stejnosměrné napětí - Direct Current
DC	označení střídy - Duty Cycle
Baud rate	přenosová rychlost v bitech za sekondu
I/O porty	vstupně výstupní porty
TxD	vysílací bit
RxD	přijímací bit
t	teplota
P	výkon [W]
U	napětí [V]
R	odpor [Ω]
P_R	výkonové zatížení odporu [W]
f_{osc}	frekvence oscilátoru
MIPS	milion instrukcí za sekundu - Million Instructions Per Second

Seznam příloh

A První příloha : zdrojový kód - vysílací strana (WinAVR)

```
;Program : Teplomer
;Datum__ : 4.5.2008
;Autor__ : David Prečan
;***************************************************************************
;
**
;******      TYP PROCESORU
*********************************************************

              .DEVICE      ATMEGA162

              .LOFUSE      0B11101111
              .HIFUSE      0B11011001

              RJMP   INIT

;***************************************************************************
;
**
;******      KONSTANTY, SYMBOLICKA JMENA
*****************************************

              .SET   XTAL = 11059200
              .SET   PCT = 1000000

              .IBIT  BSMT = PINC, 7
              .IBIT  LED1 = PORTD, 6
              .IBIT  DDR_LED1 = DDRD, 6
              .IBIT  LED2 = PORTD, 7
              .IBIT  DDR_LED2 = DDRD, 7

;***************************************************************************
;
**
;******      PROMENNE V RAM
*****************************************************

              .DSEG  SRAM_START

TXHEAD:              .BYTE  4                        ;4B
TXSIZE:              .BYTE  2                        ;2B
TXCRC1:              .BYTE  1                        ;1B
TXDATA:              .BYTE  8                        ;volitelne
TXCRC2:              .BYTE  1                        ;1B

              .CSEG

;***************************************************************************
;
**
;******      INICIALIZACE
*********************************************************

              .SECT  INIT

              LDI    ZL, 29
              LDI    ZH, 0
              ST     Z, ZH
              DEC    ZL
              BRGE   PC - 2
              LDI    ZL, 0
              LDI    ZL, LO(SRAM_START)
              LDI    ZH, HI(SRAM_START)
              LDI    XL, LO(SRAM_SIZE)
              LDI    XH, HI(SRAM_SIZE)
              LDI    R16, 0
```

```
            ST     Z+, R16
            SBIW   XL, 1
            BRNE   PC - 2



            LDI    R16, LO(RAMEND)
            OUT    SPL, R16
            LDI    R16, HI(RAMEND)
            OUT    SPH, R16
            LDI    R16, 0B00000000
            OUT    UCSR0A,      R16
            LDI    R16, 0B00011000
            OUT    UCSR0B,      R16
            LDI    R16, 0B10000110
            OUT    UCSR0C,      R16
            LDI    R16, LO(XTAL / 16 / 9600 - 1)
            OUT    UBRR0L,      R16
            LDI    R16, HI(XTAL / 16 / 9600 - 1)
            OUT    UBRR0H,      R16



            SBI    DDR_LED1
            SBI    DDR_LED2
            SBI    LED2
            RJMP   MAIN

            .ENDSECT

;*****************************************************************************
;*
;*********************   PODPROGRAMY
******************************************

            .SECT  MER_SMT

            CLR    R16
            CLR    R17
            CLR    R18
            CLR    R19


            LDI    R20, BYTE1(PCT)
            LDI    R21, BYTE2(PCT)
            LDI    R22, BYTE3(PCT)
            LDI    R23, BYTE4(PCT)

            CLR    R0
LOOP:       CLC
            SBIC   BSMT
            SEC

            ADC    R16, R0
            ADC    R17, R0
            ADC    R18, R0
            ADC    R19, R0

            SUBI   R20, 1
            SBCI   R21, 0
            SBCI   R22, 0
            SBCI   R23, 0
            BRNE   LOOP

            STS    TXDATA + 0, R16
            STS    TXDATA + 1, R17
            STS    TXDATA + 2, R18
            STS    TXDATA + 3, R19
```

```
            LDI    R16, BYTE1(PCT)
            STS    TXDATA + 4, R16
            LDI    R16, BYTE2(PCT)
            STS    TXDATA + 5, R16
            LDI    R16, BYTE3(PCT)
            STS    TXDATA + 6, R16
            LDI    R16, BYTE4(PCT)
            STS    TXDATA + 7, R16


            RET

            .ENDSECT

;*****************************************************************************
**

            .SECT  ODESLI

;tvorba      hlavičky "HDTX" (přepíše znaky z HEADSTdo TXHEAD)

            LDI    R16, TXSIZE - TXHEAD


            LDI    YL, LO(TXHEAD)
            LDI    YH, HI(TXHEAD)

            LDI    ZL, LO(HEADST *2)
            LDI    ZH, HI(HEADST *2)
LOOP1:      LPM    R18, Z+
            ST     Y+, R18

            SUBI   R16, 1
            BRNE   LOOP1

;vypočet kontrolní velikosti dat

            LDI    R16, LO(TXCRC2 - TXDATA)
            LDI    R17, HI(TXCRC2 - TXDATA)


            STS    TXSIZE + 0, R16
            STS    TXSIZE + 1, R17

;vypocitat CRC hlavicky

            EOR    R18, R18

            LDI    R16, TXCRC1 - TXHEAD

            LDI    ZL, LO(TXHEAD)
            LDI    ZH, HI(TXHEAD)
LOOP2:      LD     R19, Z+
            ADD    R18, R19
            SUBI   R16, 1
            BRNE   LOOP2

            STS    TXCRC1,      R18

;vypocitat CRC dat

            EOR    R18, R18

            LDI    R16, TXCRC2 - TXDATA

            LDI    ZL, LO(TXDATA)
            LDI    ZH, HI(TXDATA)
```

```
LOOP3:      LD     R19, Z+
            ADD    R18, R19

            SUBI   R16, 1
            BRNE   LOOP3

            STS    TXCRC2,      R18

;vyšleme celou datovou strukturu

            LDI    R16, TXCRC2 - TXHEAD + 1

            LDI    ZL, LO(TXHEAD)
            LDI    ZH, HI(TXHEAD)
LOOP4:      LD     R18, Z+
            RCALL  SENDBT
            SUBI   R16, 1
            BRNE   LOOP4

            RET

;vyslani bytu

SENDBT:             OUT    UDR0, R18
            SBIS   UCSR0A,      TXC
            RJMP   PC - 1

            SBI    UCSR0A,      TXC
            RET

;identifikacni retezec vysilani

HEADST:             .DB    'HDTX'


            RET

            .ENDSECT

;*****************************************************************************
**

            .SECT  BLIK

            SBI    LED1
            LDI    R18, 128
            LDI    R17, 187
            LDI    R16, 76
            DEC    R16
            BRNE   PC - 1
            DEC    R17
            BRNE   PC - 4
            DEC    R18
            BRNE   PC - 7
            CBI    LED1
            RET

            .ENDSECT

;*****************************************************************************
**
;******      HLAVNI PROGRAMOVA SMYCKA
*******************************************

            .SECT  MAIN

LOOP:  RCALL  MER_SMT
            RCALL  ODESLI
```

```
            RCALL  BLIK


            RJMP   LOOP

            .ENDSECT
;*************************************************************************
**
;******      KONEC PROGRAMU
*********************************************************
```

B Druhá příloha : zdrojový kód - přijímací strana (Delphi)

B.1 První část druhé přílohy : programová část Main

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, XPortClient;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Button1: TButton;
    Label1: TLabel;
    Bevel1: TBevel;
    Shape1: TShape;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    FClient: TXPortClient;
    procedure XPortCltConnect(Sender: TObject);
    procedure XPortCltDisconnect(Sender: TObject);
    procedure XPortCltError(Sender: TObject; ErrorCode: integer);
    procedure XPortCltRecv(Sender: TObject; Data: pointer; Len: word);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

//****************************************************************************

const
  XPORT_SSIGN = 'HDRX';
  XPORT_RSIGN = 'HDTX';
  SCaptionConnected = 'Teploměr - připojeno';
  SCaptionDisconnected = 'Teploměr - odpojeno';
  SLabel = '%.1f °C';

//****************************************************************************

procedure TForm1.Button1Click(Sender: TObject);
begin
  FClient := TXPortClient.Create(true, StrToInt(Edit2.Text), Edit1.Text,
XPORT_SSIGN, XPORT_RSIGN);
  with FClient do
  begin
    OnConnect := XPortCltConnect;
    OnDisconnect := XPortCltDisconnect;
    OnError := XPortCltError;
    OnRecv := XPortCltRecv;

    FreeOnTerminate := true;
    Resume;
  end;
  Button1.Enabled := false;
```

```
  Edit1.Enabled := false;
  Edit2.Enabled := false;
end;

//***************************************************************************

procedure TForm1.XPortCltConnect(Sender: TObject);
begin
  Caption := SCaptionConnected;
end;

//***************************************************************************

procedure TForm1.XPortCltDisconnect(Sender: TObject);
begin
  Caption := SCaptionDisconnected;
  Button1.Enabled := true;
  Edit1.Enabled := true;
  Edit2.Enabled := true;
end;

//***************************************************************************

procedure TForm1.XPortCltError(Sender: TObject; ErrorCode: integer);
begin
  ShowMessage(SysErrorMessage(ErrorCode));
  Caption := SCaptionDisconnected;
  Button1.Enabled := true;
  Edit1.Enabled := true;
  Edit2.Enabled := true;
end;

//***************************************************************************

procedure TForm1.XPortCltRecv(Sender: TObject; Data: pointer; Len: word);
var
  a: array[0..1] of integer;
  t: real;

begin
  if Len <> SizeOf(a) then
    Exit;
  Move(Data^, a, SizeOf(a));

  t := ((a[0] / a[1]) - 0.32) / 0.0047;
  t := t - 2.5;

  Label1.Caption := Format(SLabel, [t]);

  Shape1.Brush.Color := Shape1.Brush.Color xor $FFFF;
 /windows.beep(1000,100);
end;

//***************************************************************************

procedure TForm1.FormCreate(Sender: TObject);
begin
  Caption := SCaptionDisconnected;
end;

end.
```

B.2 Druhá část druhé přílohy : programová část XPortClient

```
unit XPortClient;

interface

uses
  Classes, SysUtils, Winsock;

type
  TXPortEventType = (xeConnect, xeDisconnect, xeError, xeRecv);
  TXPortGenericEvent = procedure(Sender: TObject) of object;
  TXPortRecvEvent = procedure(Sender: TObject; Data: pointer; DataLen: word)
of object;
  TXPortErrorEvent = procedure(Sender: TObject; ErrorCode: integer) of object;

  TXPortSignature = array[0..3] of char;
  TXPortHeader = packed record
    case integer of
      0: (Signature: TXPortSignature;
        DataLen: word;
        CRC: byte
        );
      1: (Raw: array[0..6] of byte);
  end;

  PXPortDataArray = ^TXPortDataArray;
  TXPortDataArray = array[0..0] of char;
  TXPortRecvStatus = (rsHeader, rsData, rsDataCRC);

  TXPortClient = class(TThread)
  private
    FSocket: TSocket;
    FPort: word;
    FHost: string;
    FSendSign: TXPortSignature;
    FRecvSign: TXPortSignature;
    FErrorCode: integer;
    FEvent: TXPortEventType;
    FDataPtr: pointer;
    FDataSize: word;
    FOnRecv: TXPortRecvEvent;
    FOnError: TXPortErrorEvent;
    FOnConnect: TXPortGenericEvent;
    FOnDisconnect: TXPortGenericEvent;
    procedure HandleEvent;
  protected
    procedure Execute; override;
  public
    constructor Create(CreateSuspended: boolean; APort: word; AHost: string;
      ASendSign, ARecvSign: TXPortSignature);
    procedure SendData(var Buffer; Count: integer);
    property OnRecv: TXPortRecvEvent read FOnRecv write FOnRecv;
    property OnError: TXPortErrorEvent read FOnError write FOnError;
    property OnConnect: TXPortGenericEvent read FOnConnect write FOnConnect;
    property OnDisconnect: TXPortGenericEvent read FOnDisconnect write
FOnDisconnect;
  end;

implementation

//***************************************************************************


constructor TXPortClient.Create(CreateSuspended: boolean; APort: word; AHost:
string;
```

```
  ASendSign, ARecvSign: TXPortSignature);
begin
  inherited Create(CreateSuspended);

  FSocket := INVALID_SOCKET;
  FPort := APort;
  FHost := AHost;
  FSendSign := ASendSign;
  FRecvSign := ARecvSign;
end;

//****************************************************************************

procedure TXPortClient.Execute;

  function Receive(var Buffer; Count: integer): integer;
  begin
    result := recv(FSocket, Buffer, Count, 0);
    if (result = SOCKET_ERROR) then
    begin
      result := 0;
      if WSAGetLastError <> WSAEWOULDBLOCK then
        Abort;
    end;
  end;

var
  HostEnt: PHostEnt;
  Addr: TSockAddrIn;
  TimeOut: TTimeVal;
  i, j: integer;
  FDSet: TFDSet;
  b, CRC: byte;

  Header: TXPortHeader;
  DataArray: PXPortDataArray;
  DataPos: integer;
  Status: TXPortRecvStatus;

begin
  try
    {inicializace a pripojeni}
    FSocket := socket(PF_INET, SOCK_STREAM, IPPROTO_IP);
    if FSocket = INVALID_SOCKET then
      Abort;

    HostEnt := GetHostByName(PChar(FHost));
    if HostEnt = nil then
      Abort;

    with Addr do
    begin
      sin_addr := PInAddr(HostEnt^.h_addr_list^)^;
      sin_family := PF_INET;
      sin_port := htons(FPort);
    end;

    if connect(FSocket, Addr, SizeOf(Addr)) = SOCKET_ERROR then
      Abort;

    i := 1;
    if ioctlsocket(FSocket, FIONBIO, i) <> 0 then
      Abort;
    i := 0;
    setsockopt(FSocket, SOL_SOCKET, SO_OOBINLINE, @i, SizeOf(i));

    with TimeOut do
    begin
```

```
      tv_sec := 0;
      tv_usec := 500;
    end;

    Status := rsHeader;
    DataArray := nil;
    DataPos := 0;

    FEvent := xeConnect;
    Synchronize(HandleEvent);

    {hlavni smycka}
    while (not Terminated) and (FErrorCode = 0) do
    begin
      FD_Zero(FDSet);
      FD_Set(FSocket, FDSet);
      i := select(0, @FDSet, nil, nil, @TimeOut);
      if i = SOCKET_ERROR then
        Abort;
      if i = 0 then
        continue;

      i := recv(FSocket, j, SizeOf(j), MSG_PEEK);
      if i > 0 then
      begin
        case Status of
          rsHeader:
            while Receive(b, 1) > 0 do
            begin
              Move(Header.Raw[1], Header.Raw[0], SizeOf(Header) - 1);
              Header.Raw[High(Header.Raw)] := b;

              if Header.Signature = FRecvSign then
              begin
                CRC := 0;
                for i := Low(Header.Raw) to High(Header.Raw) -
SizeOf(Header.CRC) do
                  CRC := CRC + Header.Raw[i];
                if CRC = Header.CRC then
                begin
                  Status := rsData;
                  DataPos := 0;
                  ReallocMem(DataArray, Header.DataLen);
                  break;
                end;
              end;
            end;

          rsData:
            begin
              inc(DataPos, Receive(DataArray[DataPos], Header.DataLen -
DataPos));

              if DataPos = Header.DataLen then
                Status := rsDataCRC;
            end;

          rsDataCRC:
            if Receive(b, 1) = 1 then
            begin
              CRC := 0;
              for i := 0 to Header.DataLen - 1 do
                inc(CRC, ord(DataArray[i]));

              if CRC = b then
              begin
                FDataPtr := DataArray;
                FDataSize := Header.DataLen;
```

```
                FEvent := xeRecv;
                Synchronize(HandleEvent);
              end;

              Status := rsHeader;
            end;
        end;
      end
      else
        case i of
          0: break; {konec spojeni}
          SOCKET_ERROR: Abort;
        end;
    end;

  except
    on EAbort do
      FErrorCode := WSAGetLastError;

  else
    raise;
  end;

  if FErrorCode <> 0 then
    FEvent := xeError
  else
    FEvent := xeDisconnect;

  Synchronize(HandleEvent);

  CloseSocket(FSocket);
  FSocket := INVALID_SOCKET;
end;

//******************************************************************************

procedure TXPortClient.HandleEvent;
begin
  case FEvent of
    xeError:
      if Assigned(FOnError) then
        FOnError(self, FErrorCode);
    xeRecv:
      if Assigned(FOnRecv) then
        FOnRecv(self, FDataPtr, FDataSize);
    xeConnect:
      if Assigned(FOnConnect) then
        FOnConnect(self);
    xeDisconnect:
      if Assigned(FOnDisconnect) then
        FOnDisconnect(self);
  end;
end;

//******************************************************************************

var
  WSAData: TWSAData;

initialization
  WSAStartUp($101, WSAData);

finalization
  WSACleanUp;

end.
```